Ângelo Miguel Loureiro Sarmento

Estruturas de Dados Métricas Genéricas em Memória Secundária

Lisboa

2010

Universidade Nova de Lisboa
Faculdade de Ciências e Tecnologia
Departamento de Informática

Dissertação de Mestrado

Estruturas de Dados Métricas Genéricas em Memória Secundária

Ângelo Miguel Loureiro Sarmento
(aluno nº 28031)

Orientadora: Prof. Doutora Margarida Paula Neves Mamede

Trabalho apresentado no âmbito do Mestrado em Engenharia Informática, como requisito parcial para obtenção do grau de Mestre em Engenharia Informática.

1º Semestre de 2009/2010

22 de Fevereiro de 2010

Agradecimentos

Em primeiro lugar, gostaria de agradecer à Professora Margarida Mamede, pelo apoio, disponibilidade, motivação, críticas e sugestões durante a orientação desta dissertação, que enriqueceram, sem qualquer dúvida, este documento. Gostaria também de lhe agradecer pela influência na minha vida não académica devido à sua exigência, organização e excelência.

Ao Centro de Informática e Tecnologias da Informação (CITI), pela atribuição de uma Bolsa de Iniciação Cientifica que possibilitou a realização deste trabalho.

Ao Professor Luís Caires, pela gentil disponibilização de uma máquina para a realização de testes experimentais. Sem esta máquina seria extremamente difícil concluir esta dissertação nos prazos estabelecidos.

Ao corpo de docentes do Departamento de Informática da Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa, especialmente aos Professores Fernanda Barbosa e João Seco, pelo apoio prestado sempre que necessário.

Aos colegas Carlos Rodrigues e Pedro Chambel que, amavelmente, contribuíram de forma importantíssima para a concretização deste trabalho.

Aos meus amigos mais próximos, pelo companheirismo, amizade, apoio e motivação demonstrados ao longo do tempo.

À minha família mais chegada, principalmente à minha avó, Carminda Loureiro, por tudo o que fez e faz por mim desde sempre.

Agradecer a todos os que ajudaram a realizar esta dissertação não é tarefa fácil. O maior risco que se corre ao agradecer individualmente não é decidir quem incluir, mas

decidir quem não mencionar. Então, a todos os que contribuíram de forma directa ou indirecta para a realização deste trabalho, gostaria de lhes revelar a minha gratidão.

Resumo

À medida que a complexidade dos tipos de dados modernos foi crescendo, os espaços métricos tornaram-se num paradigma popular para pesquisas por similaridade. Devido aos formatos complexos dos dados (e.g. vídeos, imagens ou sons) e também à elevada quantidade de informação, é crucial poupar tempo neste tipo de pesquisas, evitando que se analisem todos os objectos da base de dados cada vez que uma procura é efectuada. O tempo dispendido está directamente relacionado com o número de cálculos de distância entre dois objectos e com o número de acessos a disco. Sendo assim, o principal objectivo de qualquer estrutura de dados métrica implementada em memória secundária é minimizar essas duas quantidades.

Neste trabalho é apresentada a Recursive Lists of Clusters 2 (RLC2), uma estrutura de dados métrica genérica, dinâmica e implementada em memória secundária. Esta estrutura é uma variante de outra estrutura de dados, a Recursive Lists of Clusters (RLC) [Mam07].

Adicionalmente, estudam-se várias estruturas de dados inseridas no mesmo âmbito que a RLC2 e apresentam-se os resultados de uma bateria de testes que comparam os seus desempenhos. Nos testes efectuados, a RLC2 revelou-se muito eficiente nas pesquisas por proximidade e muito competitiva nas inserções de objectos.

Palavras-Chave: estruturas de dados, espaços métricos, pesquisas por similaridade, implementação em memória secundária.

Abstract

As the complexity of modern data types increased, metric spaces have become a popular paradigm for similarity searches. Due to the complex data formats (e.g. videos, pictures or sounds) and also to the large amounts of information, it is crucial to save time in this type of searches, avoiding examining all objects of the database each time a search is performed. The spent time is directly related to the number of distance calculations and the number of disk accesses. Thus, the main purpose of any metric data structure implemented in secondary memory is to minimize these two amounts.

In this work the Recursive Lists of Clusters 2 (RLC2) is presented, which is a generic, dynamic, metric data structure, implemented in secondary memory. This data structure is a variant of another data structure, the Recursive Lists of Clusters (RLC) [Mam07].

In addition, several data structures in the same scope of RLC2 are studied and a battery of experimental results that compare their performances are presented. In these experiments, RLC2 proved to be very efficient in range searches and very competitive with respect to insertion of objects.

Keywords: data structures, metric spaces, similarity search, secondary memory implementation.

Índice

Lista de Figuras

Lista de Tabelas

1. Introdução

1.1 Contexto e Motivação

Hoje em dia, e cada vez mais, assistimos a um crescimento da necessidade de armazenar grandes quantidades de informação. Esta informação tem evoluído com o passar dos anos e é considerada essencial para um sem fim de aplicações, como *data mining*, bases de dados geográficas ou biologia computacional. Muita desta informação encontra-se em formatos complexos, como por exemplo fotografias, vídeos, impressões digitais ou sequências de ADN.

Quando se pesquisam bases de dados de imagens, por exemplo, é raro querer-se descobrir se uma dada imagem se encontra na base de dados (pesquisa exacta). O que se pretende, geralmente, é encontrar as imagens que mais se assemelham a uma determinada imagem (pesquisa por similaridade).

Duas das mais populares pesquisas por similaridade são as pesquisas *por proximidade*, nas quais se procuram todos os objectos cujas distâncias a um determinado objecto não excedam um certo valor, e as pesquisas *dos k vizinhos mais próximos*, nas quais se procuram os k objectos mais próximos de um dado objecto. Para se realizar este tipo de pesquisas, é necessário que se defina uma função capaz de traduzir a proximidade entre dois objectos. Essa função é denominada por *função distância*. Se a esta função juntarmos o conjunto de todos os objectos possíveis, temos o que é designado por um *espaço métrico*.

A principal preocupação associada às pesquisas por similaridade é a minimização do número de cálculos de distâncias entre objectos, pois estes cálculos são pesados. A aplicação de propriedades dos espaços métricos, principalmente a desigualdade triangular, desempenha um papel de extrema importância no que diz respeito à redução do número destes cálculos, pois é a partir desta que é possível incluir ou descartar vários objectos do resultado de uma pesquisa sem que seja necessário realizar cálculos de distâncias com eles.

Uma forma de potenciar a redução do número de cálculos de distâncias entre objectos é guardá-los em estruturas de dados que tirem partido das propriedades dos espaços métricos. As *estruturas de dados métricas* indexam os dados de modo a utilizar estas propriedades para tornar as pesquisas por similaridade mais eficientes.

Com este trabalho, pretende-se criar e estudar uma estrutura de dados métrica eficiente.

1.2 Âmbito do Trabalho

Uma estrutura de dados métrica que, depois de ser carregada, permita realizar actualizações ao seu conteúdo é classificada como *dinâmica*. Por exemplo, a Recursive Lists of Clusters (RLC) [MB07], a M-tree [CPZ97] e a SDI-tree [TZ06b] são estruturas de dados dinâmicas. Caso contrário, diz-se que a estrutura é *estática*, como a VP-tree [Yia93], a GNAT [Bri95] ou a MVP-tree [BO97].

Independentemente desta classificação, existe outra divisão entre as estruturas de dados métricas. Estas podem estar implementadas em memória central, como a OP-tree [TZ06a], a Quadtree [FB74] ou a Kd-tree [Ben75], ou em memória secundária, como, por exemplo, a Slim-tree [TTF+02], a DF-tree [TTS+02] ou a Distance Searching Index (D-Index) [DGS+03]. As estruturas de dados implementadas em memória central estão condicionadas à dimensão dos dados a indexar, enquanto que aquelas que estão implementadas em memória secundária não sofrem dessa limitação. Uma vez que os dados das estruturas implementadas em memória secundária estão guardados em ficheiro, as questões de entrada e saída são alvo de uma grande preocupação. Para além do número de cálculos de distâncias entre objectos, o desempenho das estruturas implementadas em memória secundária também é medido pelo número de leituras e escritas em ficheiro.

Existe ainda outra forma de classificar estruturas de dados métricas. Estas são consideradas *genéricas*, se aceitam qualquer função distância ou tipo de dados, ou são classificadas como *não genéricas*, se estiverem focadas para determinados tipos de dados ou funções distância. A RLC, a M-tree e a DF-tree são exemplos de estruturas genéricas, enquanto que a OP-tree e a SDI-tree são exemplos de estruturas não genéricas que estão orientadas para dados vectoriais.

O estudo realizado nesta dissertação é direccionado para estruturas de dados métricas genéricas, dinâmicas e implementadas em memória secundária, particularmente para a RLC. Esta estrutura dispõe de duas implementações distintas, uma em memória principal [Mam07] e outra em memória secundária [Rod06]. Ambas foram desenvolvidas no Departamento de Informática da Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa.

A investigação é orientada para os algoritmos de pesquisa por similaridade, para as operações de inserção e de remoção e para detalhes arquitecturais de estruturas que se enquadram no âmbito desta dissertação: a M-tree, a Slim-tree, a DF-tree, a Symmetric M-tree (SM-tree) [SS04a] e a D-Index que, tal como a RLC, são genéricas, dinâmicas e estão implementadas em memória secundária. Deve ser encontrado um equilíbrio entre as operações de inserção, remoção e pesquisa, pois uma estrutura pode ter um bom desempenho no que diz respeito a uma determinada pesquisa mas, em contrapartida, no momento de uma inserção ou remoção, pode ter um mau desempenho, limitando a sua utilidade.

1.3 Principais Contribuições

A principal contribuição desta dissertação é a proposta de uma estrutura de dados métrica genérica, dinâmica e implementada em memória secundária que se revelou muito eficiente nos testes experimentais realizados. Essa estrutura é uma variante da RLC.

Para avaliar a eficiência da variante proposta, foram realizados vários testes experimentais nessa e em outras estruturas do mesmo âmbito. Logo, outra contribuição deste trabalho é um estudo comparativo do desempenho de seis estruturas de dados métricas.

Finalmente, foi realizada uma análise à forma final da estrutura de dados proposta, utilizando diferentes espaços métricos, tendo-se ampliado o conhecimento sobre as estruturas de dados da família da RLC.

1.4 Estrutura do Documento

Este documento está organizado em sete capítulos. Neste primeiro capítulo é apresentado o tema deste trabalho, o seu contexto, o seu âmbito e as suas principais contribuições.

No segundo capítulo está presente o trabalho relacionado. Em primeiro lugar são introduzidas algumas definições básicas necessárias à compreensão deste trabalho e, seguidamente, são apresentadas algumas estruturas de dados métricas genéricas, dinâmicas e implementadas em memória secundária. Obviamente, a RLC é merecedora de uma análise mais profunda.

No capítulo 3 são apresentadas duas estruturas de dados métricas, a Recursive Lists of Clusters 0 (RLC0) e a Recursive Lists of Clusters 2 (RLC2). A RLC0 consiste na versão original da RLC, enquanto que a RLC2 é a nova variante da RLC proposta nesta tese.

O quarto capítulo dedica-se à descrição e análise dos espaços métricos que foram utilizados na fase de testes.

No capítulo 5 encontram-se os resultados experimentais. Este capítulo começa com a parametrização das estruturas de dados testadas e, só depois, é apresentada uma análise dos resultados experimentais. A análise da RLC2 é mais exaustiva.

O capítulo 6 apresenta as conclusões extraídas da elaboração deste trabalho. É feita uma apreciação crítica ao trabalho desenvolvido e são apresentadas algumas sugestões que podem ser tidas em conta no trabalho a realizar no futuro.

No sétimo capítulo está presente a bibliografia consultada durante a execução deste trabalho.

Por fim, em anexo, encontram-se alguns resultados experimentais que não se enquadram no quinto capitulo mas que são igualmente importantes.

2. Trabalho Relacionado

2.1 Definições Básicas

Para a melhor compreensão deste documento, é seguidamente apresentada a definição de espaço métrico, bem como as principais operações realizadas em estruturas de dados métricas.

2.1.1 Espaço Métrico

No contexto desta dissertação, torna-se fundamental o conceito matemático de *espaço métrico*, que consiste num conjunto X munido de uma *métrica* (ou *distância*), isto é, de uma função $d : X \times X \rightarrow \mathbb{R}$ tal que, para quaisquer $x, y, z \in X$:

$d(x, y)$ é um número real, não negativo (não negatividade)

$d(x, y) = 0 \Leftrightarrow x = y$ (identidade)

$d(x, y) = d(y, x)$ (simetria)

$d(x, z) \leq d(x, y) + d(y, z)$ (desigualdade triangular).

A representação da semelhança ou da proximidade entre dois *pontos* (ou *objectos*) do *universo* X (conjunto de todos os pontos) a partir de uma métrica permitiu que se pudessem desenvolver estruturas de dados que podem armazenar e indexar dados de formatos complexos. Assim, temos que *estruturas de dados métricas* são estruturas de dados que usufruem das propriedades dos espaços métricos para realizarem pesquisas numa *base de dados* (subconjunto do universo) de forma mais eficiente.

A propriedade da desigualdade triangular assume um papel fundamental na redução do número de cálculos associados à função distância, pois é a partir dela que se podem descartar grandes quantidades de informação aquando das pesquisas e, dessa forma, acelerar o tempo de resposta dessas pesquisas, evitando que se analisem todos os objectos da base de dados. Tendo em conta a figura 2.1, é possível tirar conclusões sobre a distância entre dois vértices do triângulo se se souberem os comprimentos das outras duas arestas pois, em qualquer triangulo, $a < b + c$, $b < a + c$ e $c < a + b$.

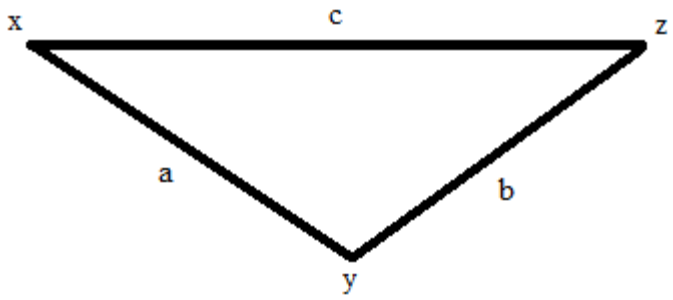

Figura 2.1: Exemplo da utilização da desigualdade triangular.

2.1.2 Estrutura de Dados Métrica

Sobre as estruturas de dados métricas foram definidas algumas operações de pesquisa, das quais se destacam estas três:

1 – *Pesquisa por proximidade* – dado um *ponto de pesquisa* p e um *raio* (real não negativo), encontrar todos os pontos na estrutura cuja distância a p seja igual ou inferior ao raio. No caso desta pesquisa, considera-se como *interrogação* o par (p, raio);

2 – *Pesquisa exacta* – verificar se um ponto está contido na estrutura. Os resultados desta pesquisa poderão ser obtidos a partir de uma pesquisa por proximidade com raio zero. Desta forma, o resultado obtido será o ponto dado na interrogação ou nenhum;

3 – *Pesquisa dos k vizinhos mais próximos* – dado um ponto de pesquisa p e um inteiro positivo k menor ou igual que o número de pontos na estrutura, encontrar um conjunto de k pontos cujas distâncias a p são iguais ou inferiores às distâncias entre p e

qualquer um dos outros pontos da estrutura. Entende-se como *interrogação*, no contexto desta pesquisa, o par (p,k).

A pesquisa dos k vizinhos mais próximos pode ser implementada a partir da pesquisa por proximidade, com o auxílio de uma estrutura de dados [CNB+01]. A estrutura auxiliar usada é, geralmente, uma fila com prioridade de dimensão k, que irá conter os objectos que, até ao momento, pertencem ao resultado final. A prioridade dos objectos que se encontram na fila é a distância entre eles e o ponto p. Inicialmente a fila começa vazia e os primeiros k elementos são adicionados a esta. De seguida, utiliza-se a pesquisa por proximidade usando como raio a maior distância entre o ponto p e qualquer ponto contido na fila. Sempre que um novo ponto é encontrado (um ponto com distância a p menor que o raio que está a ser usado), é adicionado à fila, removendo-se aquele que lá se encontrava e que tinha maior distância ao ponto p. Seguidamente, a pesquisa dos k vizinhos mais próximos continua a iterar a estrutura principal, mas agora o raio a usar na pesquisa tem de ser actualizado, pois o ponto que anteriormente era usado como referência para o raio foi excluído da fila.

Para além destas três operações, existem ainda operações de inserção e de remoção de objectos. No que diz respeito a estas duas operações, é importante fazer a distinção entre estruturas de dados métricas estáticas e dinâmicas. As estruturas de dados métricas *estáticas* são aquelas que, depois da sua construção, não admitem inserções nem remoções de objectos sem que seja necessária uma reconstrução completa. As estruturas de dados métricas *dinâmicas* permitem a inserção ou a remoção de objectos *a posteriori* da sua construção. Uma forma de construir estas estruturas, em contraste com as estruturas estáticas, é inserir, um a um, todos os objectos da base de dados à estrutura. Entre as estruturas de dados estáticas, temos, por exemplo, a VP-tree [Yia93], a GNAT [Bri95] ou a MVP-tree [BO97], enquanto que, por exemplo, a Recursive Lists of Clusters [MB07], a M-tree [CPZ97] ou a SDI-tree [TZ06b] são estruturas de dados dinâmicas.

2.1.3 Aplicações da Regra da Desigualdade Triangular

Se considerarmos em concreto o caso da pesquisa por proximidade, em que são utilizados um ponto p e um raio r, será possível, a partir da regra de desigualdade

triangular, descartar alguns objectos do domínio, sem ser necessário calcular a distância entre eles e o ponto p.

Imaginemos que, para além do ponto p e do raio r, temos mais dois pontos, x e y, que conhecemos à partida a distância entre eles (d(x,y)) e que se verifica que $d(x,y) < d(p,y) - r$. Consequentemente, por desigualdade triangular, podemos substituir na inequação anterior d(p,y) por d(p,x) + d(x,y), ficando com $d(x,y) < d(p,x) + d(x,y) - r$, que é equivalente a $d(p,x) > r$. Neste caso, podemos descartar o ponto x do resultado da pesquisa por proximidade porque, a partir de d(x,y), d(p,y) e r, podemos determinar que $d(p,x) > r$.

Seguindo um raciocínio análogo, podemos usar esta regra para incluir determinados pontos ao resultado sem efectuar cálculos de distâncias. Se considerarmos novamente os pontos x e y, a distância entre eles (d(x,y)) e soubermos que se verifica $d(x,y) \leq -(d(p,y) - r)$, podemos chegar, por desigualdade triangular, a $d(p,x) \leq r$, incluindo imediatamente o ponto x no resultado.

Se generalizarmos estes casos e considerarmos que já se encontram pré calculadas as distâncias de todos os pontos da base de dados a y, basta-nos calcular d(p,y) para podermos eventualmente descartar ou incluir vários pontos no resultado da pesquisa, poupando assim vários cálculos. Para aqueles pontos que não forem automaticamente aceites ou rejeitados por este método, terão mesmo de se efectuar os respectivos cálculos de distâncias a p e, dependendo dessa distância, esses pontos serão incluídos ou descartados do resultado da pesquisa.

2.2 Estruturas de Dados Métricas

Uma problemática associada às estruturas de dados para indexação em espaços métricos está relacionada com o facto de uma percentagem significativa delas estar implementada em memória central [Rod06], o que restringe a dimensão dos dados a indexar. Este facto constitui uma limitação dessas estruturas. Entre as estruturas de dados implementadas em memória central, temos, por exemplo, a OP-tree [TZ06a], a Quadtree [FB74] ou a Kd-tree [Ben75].

Independentemente de estarem ou não implementadas em memória central, existem estruturas de dados que estão direccionadas para determinados tipos de dados ou funções de distância. Essas estruturas são chamadas de *não genéricas*. Em oposição, as estruturas que permitem qualquer tipo de dados ou função de distância são consideradas *genéricas*.

Este trabalho concentra-se apenas em estruturas de dados genéricas, implementadas em memória secundária e dinâmicas. Estas estruturas têm vindo a conquistar mais atenção com o passar dos anos, pois existem necessidades crescentes de actualizar grandes quantidades de informação em função do tempo decorrido.

Algumas estruturas de dados implementadas em memória secundária e dinâmicas são a SDI-tree [TZ06b] ou a R-tree [Gut84], que não foram estudadas nesta dissertação porque não são genéricas.

Nesta secção são analisadas algumas estruturas de dados que estão directamente relacionadas com o assunto desta dissertação, nomeadamente, a M-tree [CPZ97], a Slim-tree [TTF+02], a DF-tree [TTS+02], a Symmetric M-tree [SS04a], a Distance Searching Index [DGS+03] e, principalmente, a Recursive Lists of Clusters [MB07]. Encontram-se abaixo as principais ilações extraídas da investigação de cada uma destas estruturas.

2.2.1 M-tree

A M-tree [CPZ97] é uma árvore que tem semelhanças com as árvores B+ [Com79]: guarda todos os objectos nas suas folhas e estas encontram-se todas ao mesmo nível. Nos nós intermédios da estrutura estão objectos denominados por objectos de encaminhamento (estes objectos são eleitos por um algoritmo de promoção). Cada *objecto de encaminhamento* está associado a um apontador para uma sub-árvore, a um raio e à distância entre si e o seu pai. Os objectos que estão contidos nessa sub-árvore encontram-se a uma distância do objecto de encaminhamento menor ou igual ao raio. É ainda de referir que se entende por *pai* de um objecto de encaminhamento o objecto de encaminhamento que referencia o nó onde este objecto se encontra. Os únicos objectos que não possuem todos estes atributos são, por razões óbvias, os que se encontram na raiz da árvore, porque não têm pai, e os que se encontram nas folhas, porque só possuem a distância para o seu pai e não dispõem de sub-árvores.

Pesquisas por Proximidade

Esta pesquisa é iniciada na raiz e percorre recursivamente todas as sub-árvores que não podem ser descartadas por desigualdade triangular. Este facto poupa cerca de 40% de cálculos de distâncias [CPZ97].

Ao chegar às folhas, aplica-se a desigualdade triangular para descartar objectos, tendo de se proceder ao cálculo das distâncias para todos os objectos que não tenham sido descartados até ao momento.

Pesquisas dos k Vizinhos Mais Próximos

Esta procura usa duas estruturas auxiliares, uma fila com prioridade (FP1) para guardar os elementos do resultado final e outra fila com prioridade (FP2) para determinar a ordem pela qual a estrutura vai ser processada.

A fila com prioridade FP2 possui apontadores para sub-árvores com elementos elegíveis para o resultado final e está ordenada pelo valor da expressão $\max(d(oe,q) - oe_r, 0)$, sendo que oe representa o objecto de encaminhamento de uma sub-árvore, oe_r o raio dessa sub-árvore e q o ponto de pesquisa. A fila está assim ordenada para que os objectos que se encontram mais perto sejam os primeiros a ser examinados e assim melhorar a eficiência.

A partir deste momento, a pesquisa será conduzida como já foi referido anteriormente, adicionando os pontos encontrados à FP1 e variando o raio da pesquisa cada vez que um novo ponto é encontrado.

Inserções

Este processo começa descendo recursivamente na árvore para inserir o objecto na folha mais apropriada mas, se a folha estiver cheia, então é desencadeada uma divisão da folha em causa.

O método para eleger a folha apropriada para a inserção consiste na descida em cada nível da árvore, escolhendo uma sub-árvore que não necessite que o seu raio seja aumentado. Se existir mais do que uma sub-árvore que reúna essas condições, então aquela cujo objecto de encaminhamento estiver mais próximo do objecto a inserir é que "fica" com o objecto. Se não existirem árvores nestas condições, então o objecto é inserido na sub-árvore cujo raio sofrerá o menor aumento de modo a inserir o objecto pretendido.

Como referido anteriormente, uma inserção de um objecto pode originar a divisão de uma folha. Assim, são necessárias medidas para lidar com essa possibilidade. A divisão de uma folha onde não cabe o objecto a inserir é feita dividindo-a em duas e distribuindo todos os objectos (os da folha antiga e o novo), por essas duas folhas. Este método é denominado por *particionamento de um nó* e é descrito mais abaixo. Para além desta divisão, são escolhidos dois pontos, um em cada folha, para serem promovidos a objectos de encaminhamento e substituírem o antigo objecto de encaminhamento que referenciava a folha antiga. Este processo é designado por *método de promoção* e é também descrito mais à frente. Se, durante a promoção, os novos objectos de encaminhamento precisarem de ser adicionados a um nó que ultrapassaria a sua capacidade, então o processo de divisão é repetido, agora com nós intermédios. Ao dividir a raiz da árvore, a profundidade desta aumenta.

Particionamento de um Nó

Este método é utilizado para escolher a melhor distribuição dos objectos de um nó antigo, por dois nós acabados de criar. Neste contexto, considera-se que o objecto a inserir já pertence ao nó antigo. Existem duas estratégias:

- *Hiperplano generalizado*: Cada objecto do nó antigo será inserido no nó que tiver o objecto de encaminhamento mais próximo dele.
- *Equilibrado*: Enquanto existirem objectos por distribuir, são repetidos os seguintes passos:

- Atribui-se ao primeiro dos novos nós o objecto que se encontra mais perto do seu objecto de encaminhamento, removendo-o da lista de objectos a distribuir.
- Atribui-se ao segundo dos novos nós o objecto que se encontra mais perto do seu objecto de encaminhamento, removendo-o da lista de objectos a distribuir.

Método de Promoção

Este método serve para escolher os dois objectos que serão promovidos ao nível acima como objectos de encaminhamento. As seguintes cinco alternativas foram testadas pelos autores da bibliografia consultada [CPZ97].

- *m_RAD*: Promove o par de objectos para o qual, depois de particionar o conjunto de entradas, a soma dos raios é mínima.
- *mM_RAD*: Semelhante à anterior mas minimiza o máximo dos dois raios.
- *M_LB_DIST*: O objecto de encaminhamento que já existia é mantido e escolhe-se, como novo objecto de encaminhamento, o mais distante desse.
- *RANDOM*: Escolhe, de forma aleatória, dois objectos.
- *SAMPLING*: Aplica-se o algoritmo mM_RAD aos objectos de uma amostra escolhida aleatoriamente. Em [CPZ97] foram utilizadas amostras com um décimo da capacidade dos nós.

Algoritmo de Carregamento Rápido

Em 1998, Ciaccia e Patella, propuseram a primeira extensão da M-tree, com um algoritmo de carregamento rápido (*bulk loading*) [CP98]. Este algoritmo é utilizado para carregar a M-tree mais rapidamente e, dessa forma, melhorar o seu desempenho. Uma desvantagem desta alternativa é a necessidade de que todos os objectos da base de dados sejam conhecidos antecipadamente.

Em primeiro lugar, escolhe-se aleatoriamente um número *n* de objectos da base de dados que irão constituir a amostra *S*. De seguida, cada elemento da base de dados será associado ao objecto de *S* que estiver mais próximo deste, produzindo *n* subconjuntos de pontos. Seguidamente, o algoritmo de carregamento rápido é executado em cada um desses subconjuntos, obtendo-se *n* sub-árvores. Desta forma, a partir da recursividade do algoritmo, são obtidas folhas com, no máximo, *n* objectos. Finalmente, o nó da raiz é criado, as sub-árvores são ligadas a este e a árvore final é obtida.

A escolha da primeira amostra é importante para a organização da estrutura. Se for escolhido um objecto que esteja rodeado por muitos objectos, a sua sub-árvore irá ser mais profunda que a sub-árvore de um objecto que esteja rodeado por poucos objectos da base de dados. A figura 2.2, retirada de [ZAD+05], mostra um exemplo de objectos no espaço bidimensional e a estrutura que se obtém no fim da primeira fase, com o algoritmo de carregamento rápido com $n = 3$.

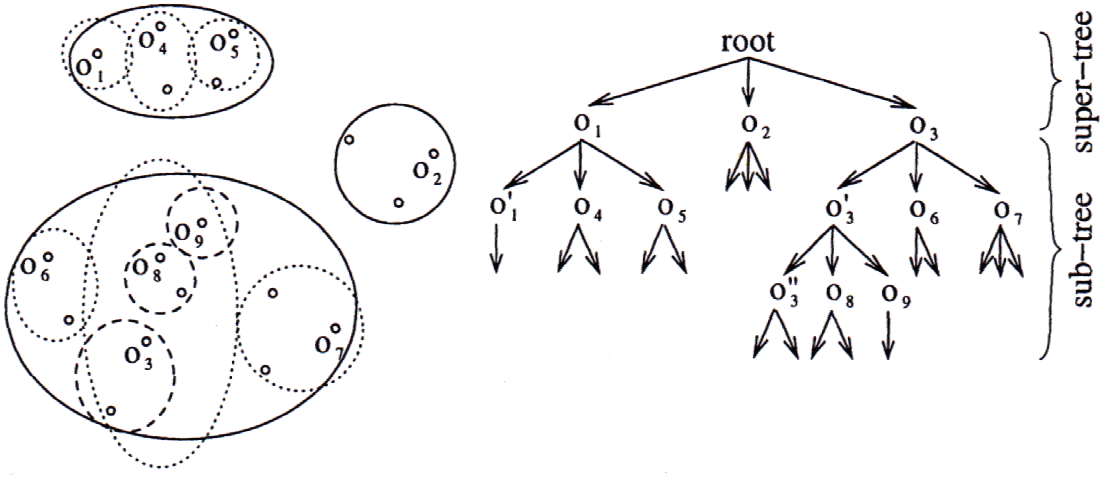

Figura 2.2: Exemplo da primeira fase do algoritmo de carregamento rápido.

No primeiro passo, o algoritmo escolhe aleatoriamente os objectos o_1, o_2 e o_3 para objectos da amostra e cria os subconjuntos de objectos que se encontram mais próximos destes. Os subconjuntos que possuem mais de três objectos são recursivamente processados e são formadas as sub-árvores.

Note-se que a árvore obtida na figura 2.2 não é equilibrada: a sub-árvore com raiz em o_2 tem menor profundidade que as restantes pois o_2 está rodeado por muito menos objectos.

O algoritmo de carregamento rápido prossegue para a próxima fase se as sub-árvores obtidas tiverem profundidades diferentes, ou seja, se a árvore não for equilibrada. As técnicas seguintes são utilizadas para resolver esse problema.

- Os nós subocupados são redistribuídos pelas outras sub-árvores e os objectos da amostra *S* que deram origem a esses nós são removidos de *S*.
- As sub-árvores mais profundas são divididas em sub-árvores menos profundas. As raízes agora obtidas irão substituir as raízes das sub-árvores mais profundas na amostra *S*.

Um nó *subocupado* é um nó que contém menos pontos que a ocupação mínima estabelecida (a partir do segundo parâmetro do algoritmo).

No exemplo da figura 2.2, os nós dos objectos o'_1 e o_9 são considerados subocupados. Estes objectos são removidos e redistribuídos pelos objectos de *S* mais próximos, o_4 e o_8, respectivamente. Como as sub-árvores com raiz em o_1 e o_3 são as mais profundas, são divididas em novas sub-árvores com raízes em o_4, o_5, o''_3, o_8, o_6 e o_7. Os objectos o_1 e o_3 são assim substituídos pelas novas raízes na amostra *S*. Finalmente, o algoritmo cria a nova árvore usando a nova amostra *S* (figura 2.3, retirada de [ZAB+05]).

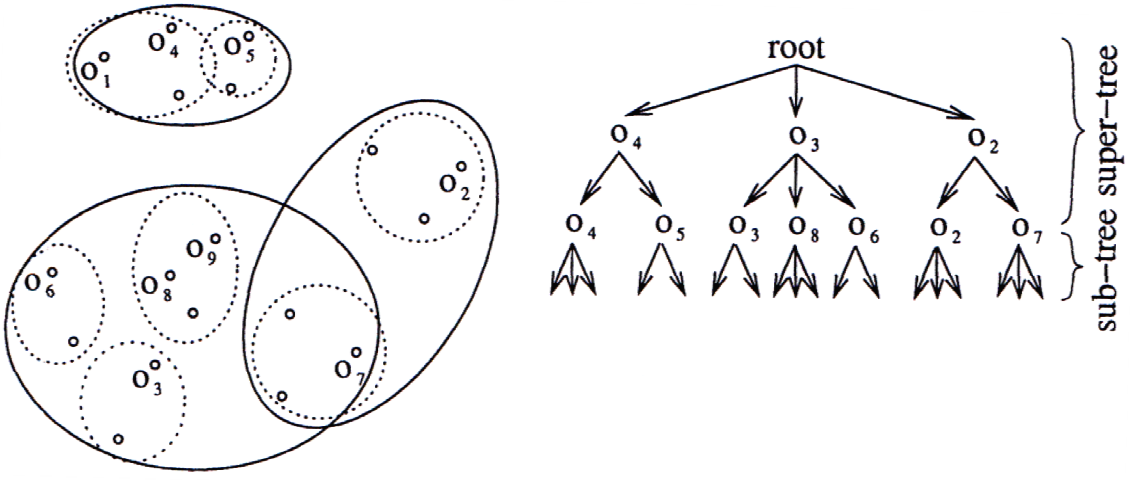

Figura 2.3: Exemplo da segunda fase do algoritmo de carregamento rápido e correspondente M-tree.

Durante a primeira fase do algoritmo, não são só calculadas distâncias entre objectos de *S* e pontos da base de dados. Também se calculam as distâncias entre os próprios elementos de *S*. Assim, a segunda fase do algoritmo faz uso da regra da desigualdade triangular, utilizando as distâncias previamente calculadas, para evitar novos cálculos de distâncias durante a redistribuição dos pontos por outros elementos de *S*.

Remoções

Até ao momento, não existem publicações relativas ao algoritmo de remoção na M-tree. Contudo, na secção 2.2.4, é descrita uma estrutura que deriva da M-tree e que introduz ligeiras modificações ao seu algoritmo de inserção. Essa alteração permitiu definir um algoritmo de remoção mais simples do que aquele que emparelharia com o presente algoritmo de inserção.

2.2.2 Slim-tree

A Slim-tree [TTF+02] é uma descendente da M-tree, que introduz três inovações que a tornam mais eficiente. Apesar destas inovações, em termos estruturais, apenas foi adicionado um campo aos objectos de encaminhamento, que indica o número de objectos da respectiva sub-árvore. As principais diferenças entre as duas estruturas de dados residem apenas em novos algoritmos.

Um desses novos algoritmos tem como objectivo melhorar o desempenho da operação de inserção, pois, tal como na M-tree, quando é encontrada mais do que uma sub-árvore em condições de abranger um objecto que está a ser inserido, terá que se decidir qual a sub-árvore em que realmente se irá inserir o objecto. Na Slim-tree existem três formas diferentes para determinar essa sub-árvore. Cada uma dessas formas será descrita posteriormente.

Adicionalmente, esta estrutura utiliza a estratégia da árvore mínima de cobertura para particionar os nós e a principal novidade é a utilização do algoritmo *Slim-down*, que torna a árvore mais estreita e rápida numa fase posterior. Como não existem diferenças

estruturais significativas entre a Slim-tree e a M-tree, só serão descritos os algoritmos acima mencionados.

Inserções

Da mesma forma que na M-tree, o algoritmo de inserção começa na raiz da árvore e tenta encontrar uma sub-árvore que consiga abranger o novo objecto. Se for encontrada mais do que uma sub-árvore em condições de o abranger, estão disponíveis três opções para o algoritmo que escolhe uma sub-árvore entre elas.

1 – *random*: Escolhe aleatoriamente uma das sub-árvores candidatas.

2 – *mindist*: Tal como na M-tree, é escolhida a sub-árvore que tem a menor distância entre o novo objecto e a raiz da sub-árvore.

3 – *minoccup*: Escolhe a sub-árvore que tem menos elementos. Esta é a opção usada por omissão, pois oferece melhor desempenho.

Se não for encontrada nenhuma sub-árvore em condições de abranger o novo ponto, é escolhida a sub-árvore cujo objecto de encaminhamento se encontra a menor distância do novo objecto.

Apesar destas diferenças na inserção de objectos, tudo o resto é igual, ou seja, quando um nó atinge a sua capacidade máxima tem de ser particionado.

Particionamento de um Nó

O algoritmo que particiona um nó constrói um grafo com todos os C objectos do nó a particionar. Mais especificamente, obtém-se um grafo não orientado e pesado, com C vértices e C * (C - 1) / 2 arcos, cujos pesos são as distâncias entre os objectos ligados pelos arcos. Em seguida, são executados os passos da figura 2.4 e ilustrados na figura 2.5 (retiradas de [TTF+02]). Note-se que a partição é efectuada com base na árvore mínima de cobertura (*MST*) do grafo.

Algorithm 1 - *Splitting of a node using MST strategy*
begin
1. Build the MST on the *C* objects of the node.
2. Delete the longest edge.
3. Report the connected components as two groups.
4. Choose the representative of each group, i.e., the object whose maximum distance to all other objects of the group is the shortest.

end

Figura 2.4: Algoritmo de particionamento de um nó usando a estratégia da árvore mínima de cobertura.

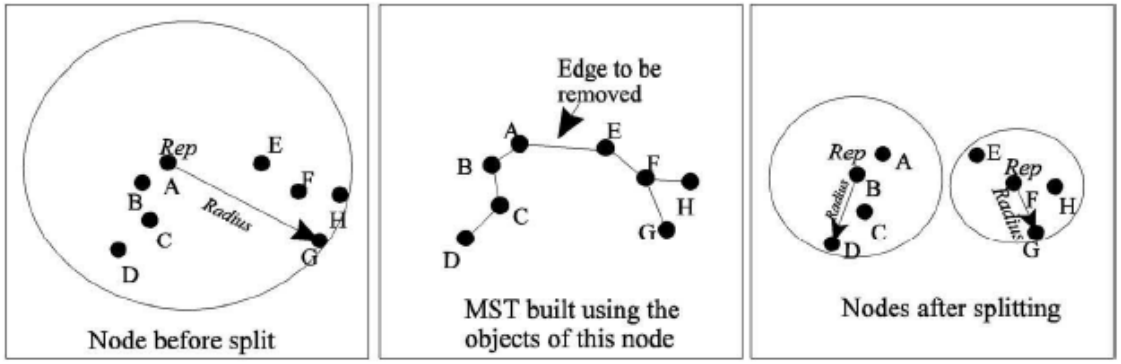

Figura 2.5: Exemplo do uso da estratégia da árvore mínima de cobertura.

Infelizmente, estes passos não garantem que cada novo nó vá receber uma percentagem razoável de objectos. Para obter uma distribuição mais equilibrada, são considerados vários arcos longos e é escolhido aquele que melhor divide os objectos do nó em dois grupos equilibrados. Se não existir nenhum (e.g. disposição em estrela), então a distribuição desequilibrada é aceite e o maior arco é removido.

O Algoritmo Slim-down

Este algoritmo (figura 2.6, retirada de [TTF+02]) tem o objectivo de minimizar as sobreposições entre os nós e, deste modo, obter melhores desempenhos nas pesquisas efectuadas. O algoritmo resolve este problema reduzindo o número de objectos que se encontram nas regiões de intersecção do mesmo nível e, também, reduzindo o número de nós existentes.

A ideia é mover pontos de uns nós para outros, que também sejam apropriados, e desta forma optimizar a estrutura, garantindo uma melhor distribuição. Depois de mover

um objecto de um nó para outro diferente, pode acontecer que o nó de origem fique sem objectos e, nesse caso, esse nó é removido. Desta maneira, para além de se reduzir a sobreposição entre os nós, também se diminui o número de nós da árvore.

Algorithm 3 - *Slim-down*
input: a built metric tree T, level h of the tree;
output: an improved metric tree T';
begin

1. For each node *i* in the level *h* of the tree, find the farthest object *c* from the representative *b*.
2. Find a sibling node *j* of *i*, that also covers object *c*. If such a node *j* exists and it is not full, remove *c* from node *i* and insert it into node *j*.
3. If node *i* is not empty then correct the radius of node *i*.
   else delete node *i*
4. Steps 1 to 3 must be applied sequentially over all nodes of a given level of the tree. If after a full round of these three steps, an object moves from one node to another, another full round from step 1 to 3 must be re-applied.

end

Figura 2.6: Algoritmo *Slim-down*.

O funcionamento do algoritmo *Slim-down* pode ser melhor observado na figura 2.7 (retirada de [TTF+02]), que mostra a organização dos objectos nos nós, antes e depois da execução do algoritmo.

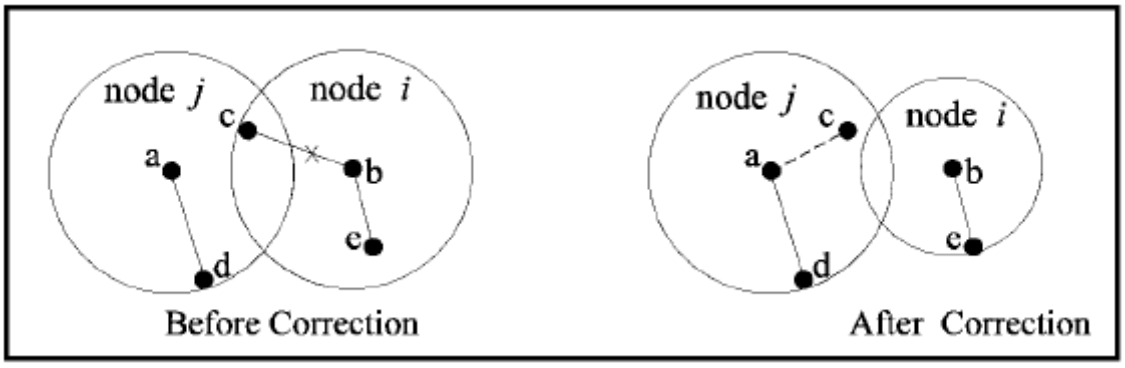

Figura 2.7: Funcionamento do algoritmo *Slim-down.*

A necessidade da utilização deste algoritmo é determinada a partir de um valor real não negativo, denominado por *relative fat-factor*, que representa, de um modo geral, uma

medida da sobreposição entre os nós do mesmo nível da árvore. Quanto maior for esse valor, maior sobreposição existe entre os nós do mesmo nível.

Apesar de existirem quatro variantes para o momento em que o algoritmo *Slim-down* é utilizado, apenas a variante em que o algoritmo é aplicado às folhas da árvore, depois desta conter todos os elementos da base de dados, foi testada em [TTF+02].

2.2.3 DF-tree

A DF-tree [TTS+02] deriva das duas estruturas anteriores, apresentando algumas diferenças importantes.

Na DF-tree é introduzido um conjunto de objectos denominados por *representantes globais*. Estes são escolhidos a partir dos pontos armazenados na árvore e permitem, como vai ser demonstrado, descartar mais objectos durante as pesquisas. Tendo em conta esta adição, é importante referir que também a estrutura dos nós da árvore sofre algumas alterações. Tanto aos objectos contidos nas folhas, como aos objectos de encaminhamento, é-lhes associado um novo campo, com um vector que contém as distâncias entre o objecto e todos os representantes globais.

A utilidade dos representantes globais é facilmente demonstrada com o auxílio da figura 2.8 (retirada de [TTS+02]). Pode observar-se que em b), usando um representante global, é possível descartar da pesquisa uma quantidade bastante maior de objectos que usando apenas um objecto de encaminhamento, como em a).

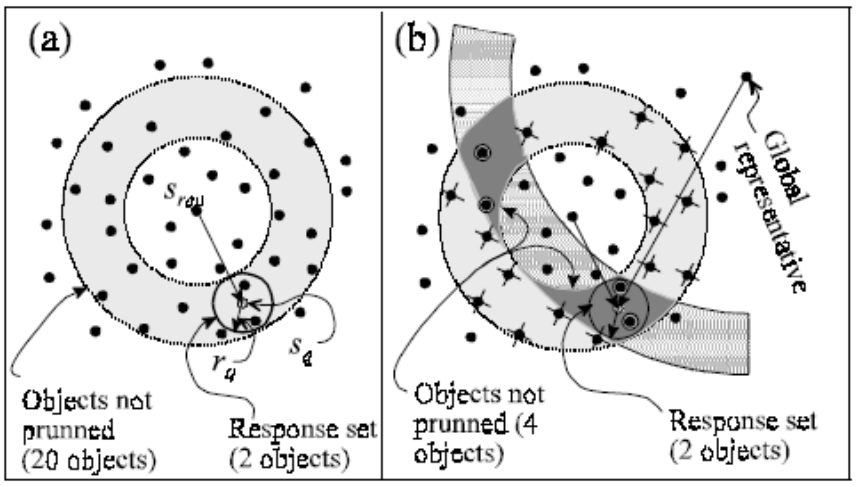

Figura 2.8: Exemplo do uso de representantes globais.

Algumas questões importantes relacionadas com os representantes globais são quando escolher o primeiro conjunto de representantes globais e quando actualizar esse mesmo conjunto.

Em relação à primeira questão, existem duas condicionantes que levam à procura de um equilíbrio entre ambas. Se a árvore ainda tiver poucos elementos, então iremos obter um mau conjunto de representantes, mas se, por outro lado, a estrutura já tiver demasiados elementos, então pode perder-se alguma "aceleração" que um bom conjunto de representantes pode proporcionar. Devido a estes factos, foi estabelecido que o primeiro conjunto de representantes globais deveria ser escolhido aquando da primeira pesquisa mas, se no momento dessa pesquisa, a árvore ainda tiver poucos elementos, então a pesquisa é efectuada sem o auxílio deste conjunto de representantes, adiando-se o momento de criação do mesmo até à execução de uma pesquisa em que a árvore possui, pelo menos, dois níveis.

Para o melhor funcionamento da estrutura, é necessário que seja feita uma manutenção adequada do conjunto de representantes globais, porque a inserção de objectos na árvore pode causar que o conjunto existente deixe de ser apropriado para o actual conjunto de objectos.

Para a escolha do momento em que o conjunto de representantes globais deve ser actualizado, é usado um algoritmo, denominado por *when to update*. Este algoritmo examina, no momento da inserção de um novo objecto, se este está ou não circunscrito pelo conjunto de representantes globais. Considera-se que um objecto está *circunscrito* por um determinado representante global, se a distância entre o objecto e esse representante for menor ou igual à maior distância entre esse representante e qualquer outro representante global. Para um objecto estar circunscrito pelo conjunto de representantes globais, terá que estar circunscrito por cada um dos representantes globais. Se o objecto não estiver circunscrito pelo conjunto dos representantes globais, o sistema atribui-lhe um peso e, quando a soma dos pesos dos objectos não circunscritos pelo conjunto dos representantes globais exceder um determinado valor, o conjunto dos representantes globais deve ser actualizado. Esse valor limite é um parâmetro da estrutura.

O peso (denotado por w) atribuído a um objecto não circunscrito é calculado de acordo com a expressão (retirada de [TTS+02]):

$$w = \sqrt[p]{\prod_{j=1}^{p} \frac{d(s_i, g_j)}{m d_j}}$$

onde p representa o número de representantes globais, s_i o objecto não circunscrito, g_j um representante global e md_j a distância máxima entre g_j e qualquer outro representante global.

Para escolher um novo conjunto de representantes globais é usado o algoritmo *how to update*, que considera como candidatos ao novo conjunto de representantes, por razões de eficiência, os elementos do presente conjunto de representantes e todos os objectos que não se encontram circunscritos por este conjunto. Considerando este conjunto de candidatos, escolhe-se um objecto ao acaso e adiciona-se ao novo conjunto de representantes o objecto que se encontra mais afastado deste. O próximo passo é adicionar ao novo conjunto de representantes o objecto que se encontra mais afastado do representante que já existe e guardar a distância entre os dois (denotada por *edge*), para auxílio na escolha dos futuros representantes. Os próximos representantes serão escolhidos a partir da seguinte expressão (retirada de [STT+01]):

$$error_i = \sum_k \left| edge - d(f_k, s_i) \right|$$

Nesta expressão considera-se f_k como um representante global e s_i como um objecto candidato a representante global que ainda não faz parte do novo conjunto de representantes. O objecto candidato que apresentar menor *error* será escolhido para representante global. O processo é repetido com o novo conjunto de representantes globais, até se obter o número desejado de representantes.

Como foi referido, a DF-tree é uma descendente da M-tree e as operações de inserção e de pesquisa dos k vizinhos mais próximos são realizadas de forma análoga às da M-tree. Por este motivo, não existirá nenhuma secção dedicada às inserções nem às pesquisas dos k vizinhos mais próximos na DF-tree.

Pesquisas por Proximidade

Estas pesquisas começam a percorrer a árvore pela raiz e usam o conjunto dos representantes globais para poder descartar sub-árvores.

As inequações 1 e 2 são avaliadas usando um objecto de encaminhamento na raiz da árvore (s_{Rep}), o objecto da interrogação (s_q), o raio da interrogação (r_q), o objecto de encaminhamento de uma sub-árvore de s_{Rep} (s_{Ri}) e o raio deste objecto de encaminhamento (r_{Ri}).

$$d(s_{Rep}, s_q)+r_q < d(s_{Rep}, s_{Ri})-r_{Ri} \quad \text{(Eq. 1)}$$

$$d(s_{Rep}, s_q)-r_q > d(s_{Rep}, s_{Ri})+r_{Ri} \quad \text{(Eq. 2)}$$

Se nenhuma das condições for verdadeira, então um dos representantes globais é usado no lugar de s_{Rep} e ambas as inequações são reavaliadas. Se nenhuma das inequações for verdadeira, então o próximo representante global é usado e assim sucessivamente, até todos os representantes globais serem usados. Se alguma das inequações for verdadeira para algum representante utilizado, então a sub-árvore pode ser descartada da pesquisa. No caso de ter que se processar uma sub-árvore, devem-se repetir os passos anteriores, usando o objecto de encaminhamento (antigo s_{Ri}) desta sub-árvore como s_{Rep}, o objecto de encaminhamento de uma sub-árvore referenciada por ele como s_{Ri} e o raio desta sub-árvore como r_{Ri}. Chegando às folhas, aplicam-se as mesmas expressões mas com r_{Ri} a zero. Se o objecto não for descartado da pesquisa por este método, terá que se proceder ao cálculo da distância entre ele e o objecto da interrogação, para se determinar se o objecto é incluído, ou não, no resultado final.

2.2.4 Symmetric M-tree

A Symmetric M-tree (SM-tree) [SS04a] é uma estrutura de dados idêntica à M-tree, que introduz duas inovações. A primeira destas inovações é a modificação do algoritmo de inserção e a outra é a definição do algoritmo de remoção. As duas inovações estão

directamente relacionadas, até porque a modificação do algoritmo de inserção foi realizada para facilitar a definição da operação de remoção.

Na opinião dos autores, o principal factor que torna a operação de remoção na M-tree "não trivial" é o facto de, depois de uma inserção, nem sempre o raio de um objecto de encaminhamento depender directamente dos seus filhos imediatos e dos respectivos raios. Na figura 2.9 (retirada de [SS04a]), é possível observar um caso em que estas dependências são violadas.

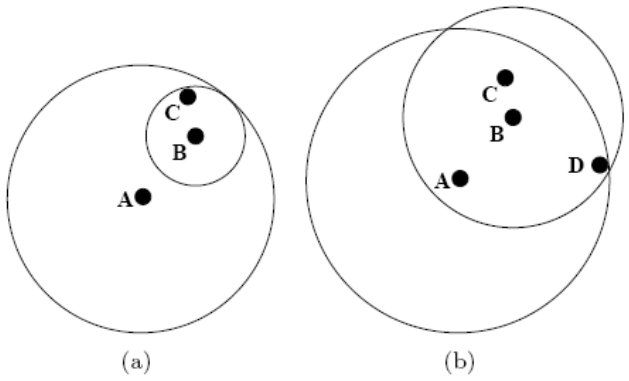

Figura 2.9: Efeitos da inserção de um ponto (D) nos raios dos nós da M-tree.

Nesta figura, durante a inserção do ponto D, o raio de A aumenta o mínimo necessário para abranger o ponto D. O mesmo acontece para a sub-árvore cuja raiz é B. É devido a esta definição, vinda da M-tree, que os autores falam em assimetria entre as operações de inserção e de remoção: na inserção, os raios das sub-árvores podem aumentar imediatamente mas, na remoção, os raios das sub-árvores não podem diminuir sem se consultar todos os objectos guardados nas folhas.

Inserções

Neste algoritmo, quando não existe uma sub-árvore apropriada para acomodar o novo objecto, em vez de ser escolhida a sub-árvore que teria o menor aumento do raio (como na M-tree), optou-se por escolher a sub-árvore que possui o objecto de encaminhamento mais próximo do objecto a ser inserido (como na Slim-tree). Esta escolha

é feita porque, nestas árvores, o raio de uma sub-árvore só depende dos seus filhos imediatos e respectivos raios. Logo, não se sabe à partida qual vai ser o real aumento do raio de uma sub-árvore.

Assim, não se procede ao aumento dos raios das sub-árvores até se encontrar a folha que vai receber o ponto. Quando este for guardado numa folha, é realizado o aumento do raio dessa sub-árvore. O novo raio é propagado para o seu pai, este actualiza o seu raio, propaga-o para o seu pai e assim sucessivamente [SS04b]. Desta forma, o raio de um objecto de encaminhamento irá sempre depender directamente dos seus filhos imediatos e dos respectivos raios.

Remoções

Com o novo algoritmo de inserção, foi possível conceber um algoritmo de remoção com funcionamento análogo. Para se encontrar o objecto que se deseja remover é realizada uma pesquisa por proximidade utilizando esse mesmo ponto como objecto de pesquisa e o valor zero como raio (pesquisa exacta), procedendo-se, seguidamente, à sua remoção. Como resultado desta operação será propagado, para o nó superior, o valor correspondente ao raio da sub-árvore, pois este pode ter diminuído com a remoção do objecto. Seguidamente, o nó superior actualiza o seu raio, propaga o respectivo raio para o seu pai e o processo repete-se até à raiz.

Nesta estrutura existe o conceito de limite mínimo de objectos num nó, podendo este ser violado com esta operação. Se, durante a remoção, um nó ficar com menos de 40% de objectos relativamente à sua capacidade máxima [SS04b], esse nó é removido, são devolvidos os objectos desse nó e estes serão inseridos na sub-árvore do objecto de encaminhamento correspondente ao vizinho mais próximo do pai do nó removido. Se todos os objectos couberem nesse nó, procede-se ao ajuste do raio desse nó e dos raios dos nós superiores. Se não for possível guardar todos os objectos nesse nó, serão usadas as políticas de particionamento que são utilizadas durante a inserção e que provêm da M-tree.

2.2.5 Distance Searching Index

Para melhorar o desempenho das pesquisas, na Distance Searching Index (D-Index) [DGS+03], foi adoptada uma abordagem à base de pivôs, em que são escolhidos alguns objectos da base de dados para essa função. Para todos os outros objectos (não pivôs), são guardadas todas as distâncias entre eles e os pivôs.

Outra característica desta estrutura é o uso de funções de particionamento, denominadas por funções *ρ-split*. Mais concretamente, uma função *ρ-split* de primeira ordem $s^{1,\rho}$ atribui a cada objecto do espaço métrico um de três símbolos, "0", "1", ou "–", de forma que, para diferentes objectos do domínio, x e y, se verifiquem as duas seguintes propriedades.

- *Separabilidade*: $s^{1,\rho}(x) = 0 \ \wedge \ s^{1,\rho}(y) = 1 \ \Rightarrow \ d(x, y) > 2\rho$.
- *Simetria*: $\rho 2 \geq \rho 1 \ \wedge \ s^{1,\rho 2}(x) \neq - \ \wedge \ s^{1,\rho 1}(y) = - \ \Rightarrow \ d(x, y) > \rho 2 - \rho 1$.

Para exemplificar, considere-se a função $f_{p,\alpha,\rho}(z)$, em que z é um objecto da base de dados, p é um pivô e α e ρ são parâmetros da função que representam distâncias. Esta função devolve:

- "0" se $d(z,p) \leq \alpha - \rho$;
- "1" se $d(z,p) \geq \alpha + \rho$;
- "–" se $\alpha - p < d(z,p) < \alpha + \rho$.

A função *ρ-split* pode ser generalizada ao concatenar n funções *ρ-split* de primeira ordem (com $n \geq 2$), com o propósito de obter uma função de particionamento de ordem n, que garante também as propriedades de simetria e de separabilidade. Sendo assim, e usando o exemplo anterior, iríamos concatenar os símbolos devolvidos pelas funções $f_{p1,\alpha 1,\rho}$, $f_{p2,\alpha 2,\rho}$, … , $f_{pn,\alpha n,\rho}$, onde p1, p2, … , pn são n pivôs distintos. Uma função *ρ-split* de ordem n gera, para cada objecto do domínio, uma sequência de n símbolos.

Para complemento da função *ρ-split* de ordem n, é usada outra função que traduz as cadeias de n símbolos geradas em inteiros, de modo a dividir os objectos em diferentes grupos. Quando todos os símbolos gerados para um objecto são diferentes de "–", a função de tradução interpreta esses símbolos como um número em binário, que é sempre menor que 2^n. Caso contrário, a função devolve como resultado 2^n. Desta maneira é possível

particionar os objectos do domínio em $2^n + 1$ conjuntos disjuntos, sendo que os primeiros 2^n conjuntos são denominadas por *conjuntos separáveis*, enquanto que o último conjunto se chama *conjunto de exclusão*.

A propriedade de separabilidade garante que a distância de um objecto de um conjunto separável a um objecto de outro conjunto separável é maior que 2ρ. Esta propriedade será usada em pesquisas por proximidade, uma vez que uma interrogação com raio menor ou igual a ρ necessita de aceder a apenas um dos conjuntos separáveis e, possivelmente, ao conjunto de exclusão.

Quando o parâmetro ρ muda, os conjuntos aumentam ou diminuem correspondentemente, sendo que a propriedade de simetria garante uma reacção uniforme em todos os conjuntos.

De forma a que sejam definidas as funções *ρ-split*, é necessário que exista um conjunto de pivôs. Para além disso, por motivos de desempenho, a D-Index faz uso desse conjunto de pivôs para descartar objectos dos resultados das pesquisas, tal como a DF-tree utiliza o conjunto de representantes globais.

Arquitectura de Armazenamento

A ideia principal da D-Index é criar uma estrutura de armazenamento em vários níveis, que usa várias funções *ρ-split* de ordem n, uma por cada nível, para criar um vector de contentores (*buckets*) que guardam objectos.

No primeiro nível, é usada uma função *ρ-split* para separar os objectos de todo o conjunto de dados. Para qualquer outro nível, só os objectos que seriam guardados no contentor de exclusão do nível anterior são armazenados nos contentores separáveis deste nível. Finalmente, o contentor de exclusão do último nível forma o contentor de exclusão de toda a estrutura D-Index. Assim, uma estrutura com h níveis que usa uma função *ρ-split* de ordem n em cada nível terá $h * 2^n$ conjuntos separáveis e 1 conjunto de exclusão.

É de notar que as funções *ρ-split* podem ter ordens diferentes, tipicamente decrescendo com o nível, permitindo que a estrutura possua níveis com diferentes números de contentores.

Pesquisas por Proximidade

Para uma pesquisa deste tipo, é comum aceder-se a todos os níveis da estrutura e, eventualmente, ao contentor de exclusão. No entanto, devido à propriedade de simetria, em alguns casos especiais, é possível limitar a pesquisa a apenas um contentor separável. No caso geral, a pesquisa é efectuada de acordo com o raio da interrogação.

1 – Se o raio da interrogação for inferior ou igual a ρ, existem duas possibilidades:

a) Só é necessário aceder a um contentor separável nesse nível;
b) Nenhum contentor é acedido nesse nível pois a região da interrogação está contida na zona de exclusão do nível.

Em ambos os casos, passa-se para o próximo nível ou examina-se o contentor de exclusão, se estivermos no último nível.

2 – Se o raio da interrogação for superior a ρ, será necessário aceder a mais contentores separáveis por nível e, possivelmente, consultar também o contentor de exclusão.

Pesquisas dos k Vizinhos Mais Próximos

A principal diferença deste algoritmo para os outros previamente estudados é a divisão da pesquisa em duas fases. Na primeira fase, é adoptada uma estratégia optimista e assume-se que o k-ésimo vizinho mais próximo se encontra a uma distância inferior ou igual a ρ. Assim, como nas pesquisas por proximidade, a busca é limitada a um conjunto separável por nível. Nos conjuntos inspeccionados, a pesquisa é realizada com os mesmos princípios das pesquisas dos k vizinhos mais próximos vistas anteriormente.

Se, no fim da primeira fase, o k-ésimo vizinho mais próximo se encontrar realmente a uma distância inferior ou igual a ρ, então a pesquisa termina. No caso contrário, será necessário consultar outros conjuntos separáveis, ignorando os conjuntos acedidos até á altura.

Inserções

Para realizar uma inserção, é aplicada ao objecto a função *ρ-split* de ordem n do primeiro nível e é determinado o conjunto onde inserir o objecto. Se o resultado for um conjunto separável, o objecto é aí inserido. Se o resultado for o conjunto de exclusão, então o processo repete-se no nível seguinte recorrendo-se à função *ρ-split* desse nível. Se não for encontrado nenhum conjunto separável, o objecto é guardado no contentor de exclusão da estrutura.

Remoções

Na bibliografia consultada não estava presente nenhuma descrição de como é realizada a remoção de objectos. No entanto, é referido que tal é possível. A única alusão à remoção ocorre quando os autores afirmam que o seu custo é maioritariamente composto pelo custo de uma pesquisa exacta e que este, em comparação com o da M-tree, é geralmente muito baixo.

2.2.6 Recursive Lists of Clusters

A Recursive Lists of Clusters (RLC) [Mam07] divide os objectos da base de dados em vários níveis de listas de agrupamentos (*clusters*). Cada *agrupamento* tem um objecto denominado por *centro*, o raio (que é fixo e igual para todos os agrupamentos), o nível a que se encontra, o tamanho do seu interior e o respectivo interior, sendo que o interior contém um conjunto de pontos cuja distância ao centro é inferior ou igual ao raio. Considera-se que um ponto do universo pertence à *região* de um agrupamento, se a distância do ponto ao centro do agrupamento não exceder o raio do agrupamento.

Os interiores dos agrupamentos podem estar organizados em novas listas de agrupamentos ou, se forem folhas, apresentam a forma de um vector de pontos. A capacidade das folhas é fixa e o interior de um agrupamento só terá a forma de uma folha se não exceder essa capacidade. Se, pelo contrário, o interior de um agrupamento tiver um

número de pontos superior à capacidade de uma folha, este irá apresentar a forma de uma lista de agrupamentos.

A cada ponto da estrutura está associado um vector com as distâncias a cada um dos centros dos agrupamentos em que este está contido. O vector está ordenado do centro do último agrupamento em que o ponto está contido para o centro do primeiro. Nas folhas, os pontos estão organizados decrescentemente pelo primeiro índice desse vector, ou seja, pela distância ao centro do último agrupamento em que estão contidos.

Na figura 2.10 (retirada de [Mam07]) pode-se observar uma RLC com raio ρ e com folhas de capacidade 5. A letra c representa o centro, r o raio, l o nível, s o tamanho do interior e I o interior do agrupamento.

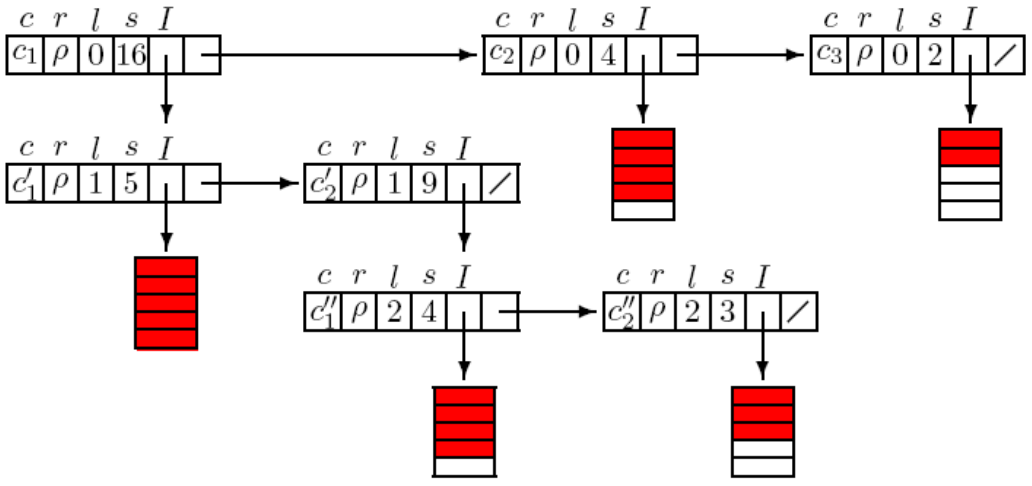

Figura 2.10: Exemplo de uma RLC.

Pesquisas por Proximidade

Esta pesquisa começa por investigar a relação entre cada agrupamento e a interrogação e, a partir daí, obtém-se um de seis possíveis resultados:

1 – A interrogação contém o centro do agrupamento e está contida no agrupamento;

2 – A interrogação contém estritamente o agrupamento;

3 – A interrogação contém o centro do agrupamento e intersecta o agrupamento sem o conter e sem estar contida no agrupamento;

4 – A interrogação não contém o centro do agrupamento e está contida no agrupamento;

5 – A interrogação não contém o centro do agrupamento e intersecta o agrupamento sem estar contida neste;

6 – A interrogação é disjunta do agrupamento.

Uma vez que esta relação entre a interrogação e o agrupamento é identificada, a pesquisa pode ter que analisar o interior do agrupamento, de modo a encontrar os pontos desejados. À medida que a pesquisa decorre, é construído um vector com as distâncias entre o objecto de pesquisa e os centros dos agrupamentos cujos interiores são analisados. Este vector é denominado por *minDists.* O vector *minDists* resulta do facto de que sempre que se chega ao interior de um agrupamento, existem dois limites para a distância entre os seus objectos e o seu centro a partir dos quais os objectos podem ser incluídos ou descartados do resultado da pesquisa.

Através do uso da regra da desigualdade triangular, do vector *minDists* e do vector com as distâncias entre cada objecto e os centros dos agrupamentos em que está contido, podem ser obtidas três respostas diferentes para cada objecto: pode ser automaticamente descartado, pode ser imediatamente adicionado à resposta ou nenhuma destas duas, o que leva a que apenas seja necessário calcular distâncias para estes últimos.

Inserções

O algoritmo de inserção começa por percorrer a lista raiz em busca de um agrupamento cujo raio seja maior ou igual à distância do seu centro ao novo ponto. Se nenhum agrupamento satisfizer estas condições, é criado um novo agrupamento no final da lista, com o novo ponto como centro.

Ao ser encontrado um agrupamento com condições para acomodar o ponto, insere-se o ponto no interior do mesmo. Se esse interior for uma lista de agrupamentos, o mesmo processo é repetido para essa lista. Se o interior for uma folha, existem duas hipóteses:

- Se a folha não estiver totalmente ocupada, o novo ponto é inserido de forma a manter a ordenação supramencionada.
- Se a folha estiver totalmente ocupada, é transformada numa lista de agrupamentos e o ponto é aí inserido.

Remoções

A remoção de um ponto, tal como a inserção, percorre recursivamente as listas de agrupamentos até encontrar um agrupamento cuja região contenha o ponto. Uma vez encontrado o ponto a remover pode verificar-se uma das seguintes situações:

- Se o ponto a remover for o centro de um agrupamento, o agrupamento é removido e os pontos contidos no seu interior serão novamente inseridos na estrutura. É de notar que novos agrupamentos podem surgir depois destas inserções.
- Se o ponto estiver numa folha, procede-se à sua remoção e actualiza-se o vector de pontos para que não existam posições vazias entre os pontos.

Depois da remoção de um ponto, sempre que uma lista de agrupamentos fique com o mesmo número de pontos que a capacidade das folhas, esta lista é imediatamente transformada numa folha.

Formato da RLC em Ficheiro

A organização da RLC em ficheiro foi concebida por Carlos Rodrigues, em [Rod06], e vem acompanhada de três regras relacionadas com a ocupação das páginas.

- Não podem existir páginas com zero elementos.
- No caso de não estarem completamente ocupadas, não podem existir quaisquer posições vazias entre os elementos presentes na página.
- Nenhum elemento poderá ocupar mais que uma página. Esta regra constitui uma limitação desta implementação em memória secundária, mas não da estrutura em si.

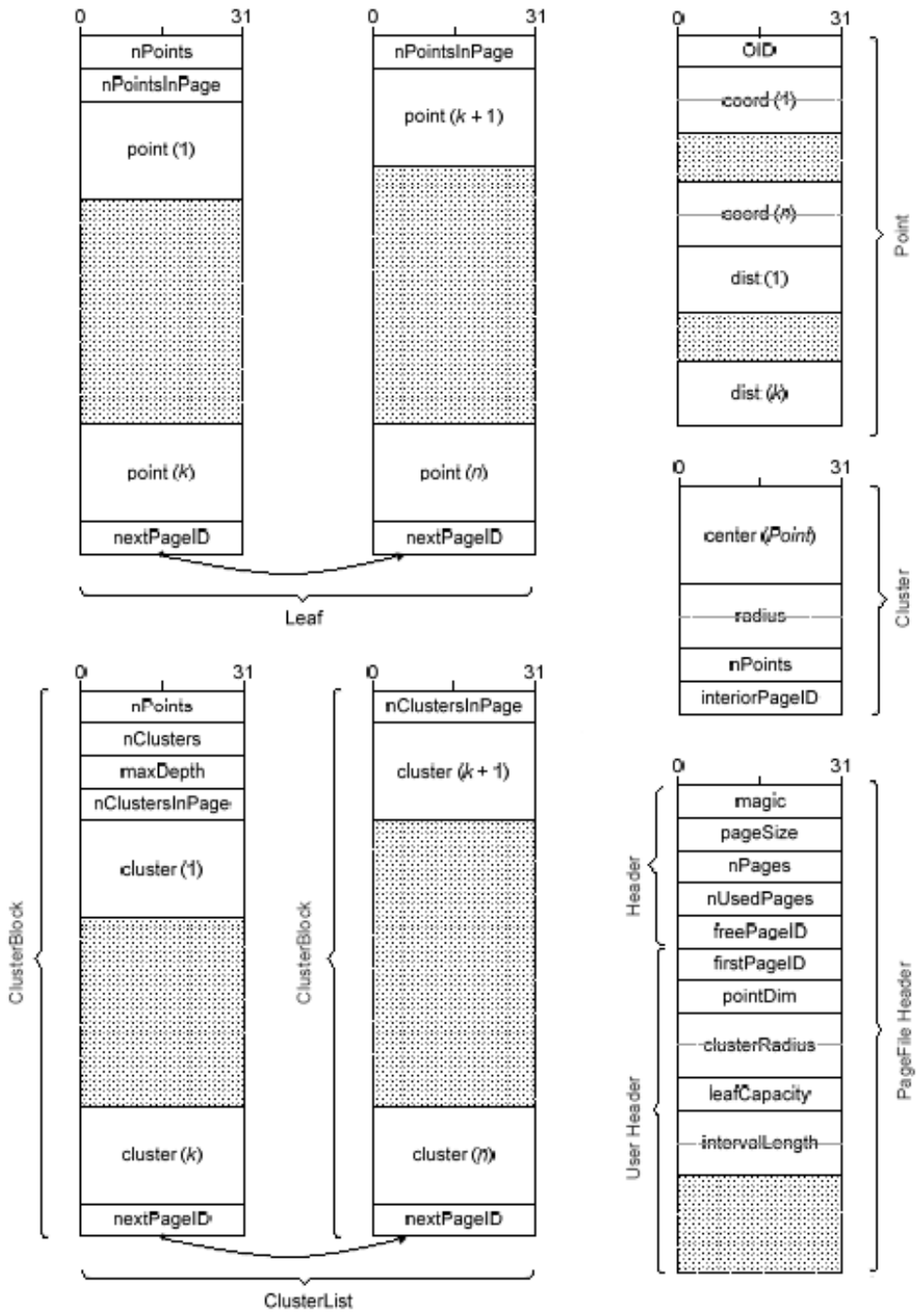

Figura 2.11: Formato em disco dos componentes da RLC.

A organização dos componentes da RLC, conforme a estrutura da figura 2.11 (retirada de [Rod06]), foi concebida de modo a minimizar o número de acessos a disco e a guardar apenas a informação necessária. A partir dessa figura, é possível observar que tanto

as listas de agrupamentos como as folhas da RLC têm a forma de sequências de páginas, em que cada página guarda parte da informação do respectivo componente. O significado de cada um dos componentes da figura 2.11 encontra-se descrito a seguir.

- *Leaf* – Representa uma folha da RLC. Na primeira página existe um campo denominado por *nPoints* que permite saber o número total de pontos de uma folha, sem que seja necessário carregar todas as páginas da respectiva folha. O campo *nPointsInPage* contém o número de pontos que estão guardados nessa mesma página e o campo *nextPageID* guarda o identificador da próxima página. Em todas as páginas de uma folha estão também guardados os pontos, na sua forma serializada, que nela estão contidos. Estes estão guardados nos campos *point*.
- *ClusterList* – Este componente representa uma lista de agrupamentos da RLC. Os campos comuns a todas as páginas de uma *ClusterList* são o *nClustersInPage*, que guarda o número de agrupamentos de uma determinada página, o *nextPageID*, com o identificador da próxima página, e os campos *cluster*. Estes últimos guardam os agrupamentos, depois de serializados, que pertencem a uma lista de agrupamentos.

  Na primeira página deste componente estão guardados o número total de pontos de uma lista de agrupamentos, o número total de agrupamentos dessa mesma lista e o número máximo de níveis atingido actualmente por essa lista. Estes valores encontram-se contidos nos campos *nPoints*, *nClusters* e *maxDepth*, respectivamente.

  O campo *maxDepth* é utilizado para dimensionar os vectores de distâncias associados aos pontos.
- *Point* – Um ponto na sua forma serializada. Nos campos *coords* estão contidas cada uma das coordenadas do ponto, enquanto que nos campos *dist* estão guardadas cada uma das distâncias entre o ponto e os centros dos agrupamentos em que está contido. O campo *OID* permite guardar uma referência para um objecto externo, mas este não é usado nesta implementação.

- *Cluster* – Representa um agrupamento serializado. O centro, o raio e o identificador da primeira página do interior do agrupamento encontram-se nos campos *center*, *radius* e *interiorID* respectivamente. O campo *nPoints* guarda o número total de pontos do agrupamento e permite que se saiba esse valor sem que se carregue o seu interior. A partir deste valor é possível saber se o interior do agrupamento é uma folha ou uma lista de agrupamentos. Se o agrupamento não tiver qualquer ponto no seu interior, o valor de *nPoints* será zero e, nesse momento, o interior não existe.
- *PageFileHeader* – Este componente está dividido em duas partes: o cabeçalho do ficheiro e os dados globais da estrutura, *Header* e *User Header*, respectivamente. Na primeira parte, o campo *magic* é um inteiro fixo que serve para verificar se o ficheiro é válido. A dimensão das páginas, em bytes, está guardada no campo *pageSize* e o campo *nPages* guarda o número total de páginas no ficheiro. O número de páginas usadas em cada momento e o identificador da primeira página da lista das páginas livres encontram-se, respectivamente, nos campos *nUsedPages* e *freePageID*. Na segunda parte encontram-se o identificador da primeira página da lista de agrupamentos raiz da RLC, *firstPageID*, a dimensão do espaço[1], *pointDim*, o raio dos agrupamentos, *clusterRadius*, a capacidade das folhas, *leafCapacity*, e o comprimento do intervalo de variação das coordenadas dos pontos, *intervalLength*. Este valor é usado apenas para verificar que os ficheiros de testes contêm pontos cujas coordenadas variam no intervalo esperado [Rod06].

[1] Considerou-se, nesta implementação, que o universo do espaço métrico tinha a forma $[a,b]^k$, onde a e b são números reais tais que $a < b$ e $k \geq 2$ é a dimensão do espaço. A métrica era a distância euclidiana.

Implementação da RLC

A RLC encontra-se implementada em C++ porque permite atingir o máximo de eficiência ao mesmo tempo que simplifica as tarefas de gestão dos dados em disco [Rod06].

Assim, a organização dos seus componentes e respectivos papéis é a seguinte.

- *Point* – Classe que representa um ponto no espaço métrico. É onde está implementada a respectiva função de distância.
- *Point::Distances* – Esta classe, membro de *Point*, contém as distâncias do respectivo ponto aos centros dos agrupamentos em que está contido.
- *PointSet* – Representa um conjunto de pontos e é usada para acumular resultados das pesquisas realizadas na RLC.
- *PointIterator* – É uma classe abstracta que representa um iterador sobre pontos.
- *Query* – Simboliza uma pesquisa por proximidade a ser realizada na RLC. Estende a classe *Point* pois uma pesquisa deste tipo é constituída por um ponto e um raio de pesquisa. Irá fazer uso da classe *Query::Distances* como limite mínimo para as distâncias aos centros dos agrupamentos.
- *Page* – Funciona como uma representação das páginas. Todas as operações de leitura ou escrita farão uso desta classe.
- *PageFile* – Responsável pela gestão das páginas. É nesta classe que são feitas as leituras e as escritas nos ficheiros, servindo as instâncias da classe *Page*.
- *Interior* – Classe abstracta que define todas as operações que um interior de um agrupamento deve suportar.
- *Leaf* – Implementa a classe abstracta *Interior* e representa uma folha da RLC, onde são guardados objectos da classe *Point*.
- *Leaf::Iterator* – Implementa a classe abstracta *PointIterator* e é usada para iterar os pontos que se encontram numa folha. Este iterador é usado, por exemplo, quando uma folha é transformada numa lista de agrupamentos e todos os pontos lá contidos necessitam de ser reinseridos na estrutura.

- *Cluster* – Representa um agrupamento da RLC. Esta classe faz uso da classe abstracta *Interior* que só é instanciada se existirem mais pontos no agrupamento para além do seu centro.
- *Cluster::Iterator* – Análogo ao *Leaf::Iterator* mas usado no contexto dos agrupamentos. É usado, por exemplo, quando um agrupamento é removido e os seus pontos necessitam de ser reinseridos na estrutura.
- *ClusterBlock* – Está directamente associado a uma página e representa um bloco de uma lista de agrupamentos. Esta classe existe para uma melhor gestão do espaço das páginas. Assim, os agrupamentos de uma lista de agrupamentos são guardados em ficheiro com a forma de blocos.
- *ClusterList* – Implementa a classe abstracta *Interior* e representa uma lista de agrupamentos.
- *ClusterList::Iterator* – Outra classe que implementa a classe abstracta *Iterator* e que é utilizada, por exemplo, na situação oposta ao *Leaf::Iterator.*
- *ClusterList::Clusters* – Funciona como iterador de agrupamentos de uma lista de agrupamentos. Recorre à classe *ClusterBlock* para inserir e remover agrupamentos.
- *Statistics* – Onde são guardadas todas as estatísticas geradas durante a execução do programa. As classes Point e PageFile fazem uso desta classe para guardar o número de cálculos de distâncias entre pontos e o número de operações de escrita e leitura. Para além de serem guardados os valores totais destas métricas, são também guardados os valores por operação de inserção, remoção e pesquisa.
- *Parameters* – Classe que encapsula os parâmetros globais da estrutura, como o raio dos agrupamentos ou a capacidade das folhas. Nunca existe mais do que uma instância desta classe durante a execução do programa.

As relações entre todas as classes acima descritas podem ser observadas através do diagrama de classes da figura 2.12.

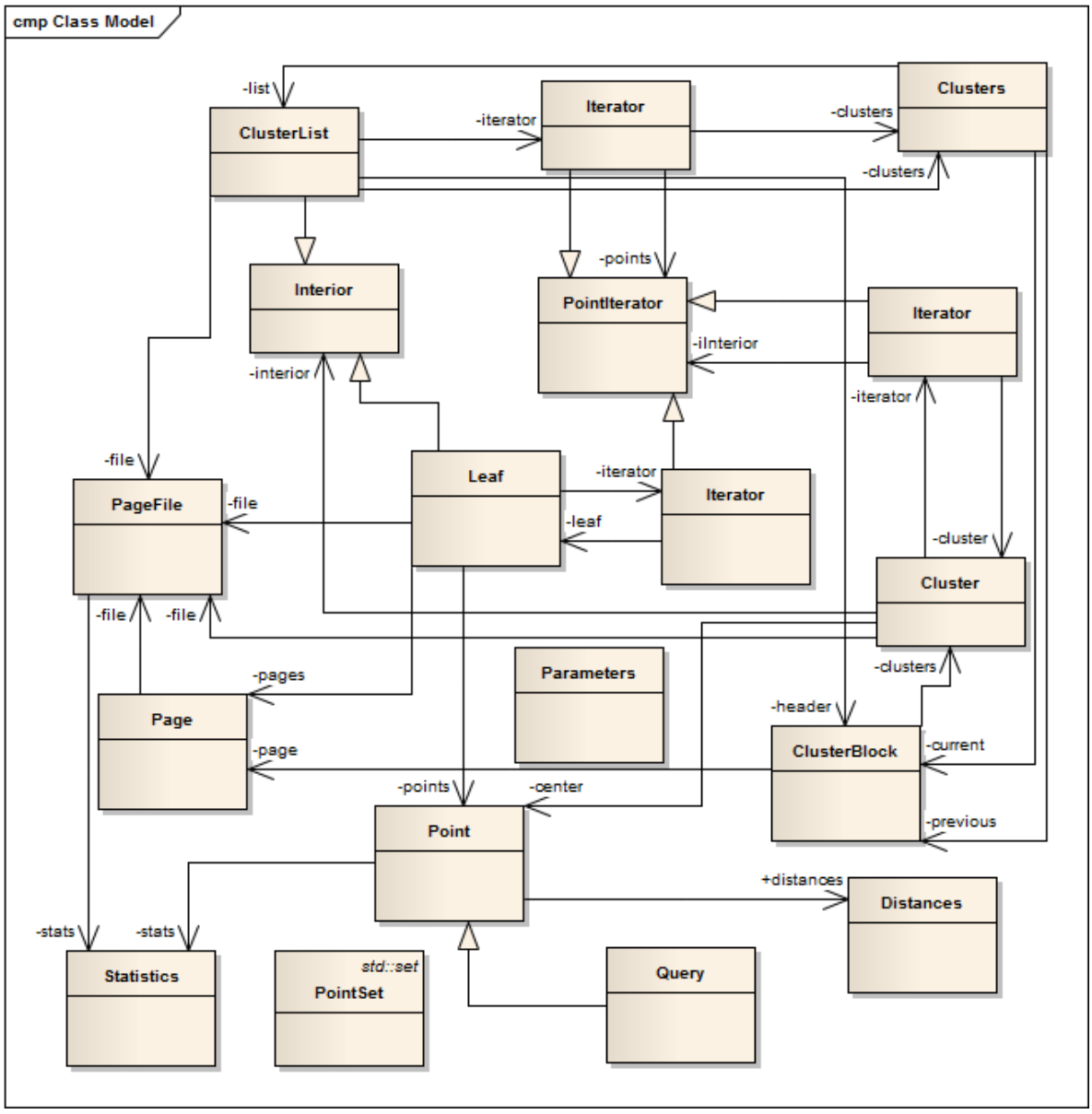

Figura 2.12: Diagrama de classes da RLC.

Alguns pormenores desta implementação são merecedores de destaque. O primeiro é o facto dos interiores dos agrupamentos só serem carregados quando necessário. As operações de inserção, remoção e pesquisa realizam muitos testes em que se acede simplesmente ao centro e ao raio dos agrupamentos, não sendo necessário consultar os seus interiores. Nestes casos, os mesmos não são carregados, poupando assim vários acessos a disco. Um acontecimento semelhante ocorre nas folhas e nas listas de agrupamentos. As

suas páginas só vão sendo lidas à medida que vão sendo necessárias. Num caso de pesquisa em que se acede a uma folha ou a uma lista de agrupamentos que ocupa mais do que uma página, se, durante a análise da primeira página for encontrada toda a informação desejada, as páginas seguintes não serão carregadas.

Outro pormenor interessante verifica-se quando dois blocos de listas de agrupamentos são fundidos num só para reduzir o número de páginas ocupadas e, por sua vez, minimizar o número de acessos a disco. Este acontecimento tem lugar sempre que a soma da ocupação de dois blocos consecutivos é igual ou inferior à capacidade de cada um.

3. Alterações à Recursive Lists of Clusters

A partir da implementação da RLC, foi possível concluir que uma das suas potenciais limitações está relacionada com o tamanho dos seus pontos. Uma vez que estes têm associados a si um vector com as distâncias aos centros dos agrupamentos em que estão contidos, os pontos que se encontram em níveis muito profundos ocuparão demasiado espaço em disco. Para além dessa limitação, a implementação da RLC não estava preparada para armazenar qualquer tipo de objectos.

Desta forma, foram necessários dois tipos de alterações à RLC. O primeiro diz respeito à sua implementação, uma vez que o âmbito deste trabalho está orientado para estruturas de dados genéricas. O segundo, a nível estrutural, foi realizado para lidar com o problema do tamanho dos pontos que se encontram em níveis muito profundos.

3.1 Recursive Lists of Clusters Genérica

A principal alteração realizada foi, na classe *Point*, a substituição do seu membro de dados, que antes era de tipo *double*,* por um membro de dados de um tipo genérico T. Com esta alteração, terá que ser desenvolvida uma classe diferente para cada espaço métrico que se queira utilizar. É nessa classe que estará presente, entre outros, o conteúdo do ponto, a função de distância e as funções de serialização e deserialização. Na figura 3.1 pode-se observar a estrutura de uma classe que representa uma palavra. Para além do que foi acima descrito, dos construtores, do destrutor e dos selectores, esta classe possui ainda um método para a escrita do seu conteúdo para um *stream* e dois métodos que retornam um booleano. O método *equals* serve comparar o conteúdo de dois pontos, enquanto que o método

isSmaller existe para que os resultados das pesquisas sejam ordenados. Existe também, neste caso, o método privado *minimum* que auxilia o cálculo da distância entre dois pontos.

```
#ifndef _WORD_POINT_H
#define _WORD_POINT_H

#include <iostream>
#include <cstring>

using namespace std;

class WordPoint {
public:
    WordPoint();
    WordPoint(const WordPoint &other);
    WordPoint(const char word[], int dim);
    virtual ~WordPoint();

    virtual int getDimension() const;
    virtual const char* getData() const;

    bool equals(const char* other);
    double distanceTo(const char* other);
    bool isSmaller(const char* other);
    void write(std::ostream& output = std::cout) const;
    const char* serialize();
    void unserialize(int dim, const char *data);

private:
    int minimum(int a, int b, int c);

    int dim;
    char *data;

};

#endif // WORD_POINT_H
```

Figura 3.1: Estrutura da classe que irá representar uma palavra.

Outra modificação que ocorreu foi a introdução de um novo parâmetro na estrutura, o *entrySize*. Na versão anterior, como todas as coordenadas de um ponto eram de tipo *double*, o número de bytes a ler de uma página, por exemplo, para construir as coordenadas de um ponto, era obtido multiplicando o tamanho de um *double* pelo número de coordenadas que o ponto tinha. Agora, com pontos genéricos, este parâmetro é necessário

para realizar essas contas. O parâmetro *entrySize* corresponde ao número de bytes que cada componente de um ponto ocupa. Se quisermos, por exemplo, continuar a usar os pontos que eram usados antes, *entrySize* terá que ter o valor oito mas, se usarmos pontos que consistem em vectores de *char's*, então *entrySize* deverá ter o valor um. Este valor é guardado junto dos outros parâmetros globais da estrutura, tanto na classe *Parameters*, como no novo campo *entrySize* que fará parte do *PageFileHeader.*

Por fim, procedeu-se à eliminação do campo *radius* do agrupamento serializado. Uma vez que o raio dos agrupamentos já está presente no *PageFileHeader,* a sua presença em dois locais distintos foi considerada redundante.

3.2 Variante à Recursive Lists of Clusters (RLC2)

Segundo a implementação da RLC, todos os agrupamentos têm o mesmo raio, independentemente do nível a que se encontram. Desta forma, quando uma folha atinge a sua capacidade máxima e é transformada numa nova lista de agrupamentos, é comum que apenas um dos agrupamentos dessa nova lista fique com a grande maioria dos pontos do agrupamento do nível acima (ver figura 3.2). É possível observar-se, nessa figura, que o interior do agrupamento com centro C consiste numa lista com dois agrupamentos. Uma vez que o raio desses agrupamentos é igual, o primeiro agrupamento a ser criado irá possuir uma área bastante maior do que a dos agrupamentos criados posteriormente. Consequentemente, esta lista de agrupamentos irá ter um comprimento bastante reduzido e a maioria dos pontos que pertencerem ao agrupamento com centro C irão pertencer também ao agrupamento com centro C1. Tendo este exemplo em conta, é possível perceber-se que, quando uma folha é transformada numa lista de agrupamentos, os pontos da folha não serão igualmente divididos pelos agrupamentos do nível abaixo. Alguns desses agrupamentos irão ter bastantes mais pontos que os outros.

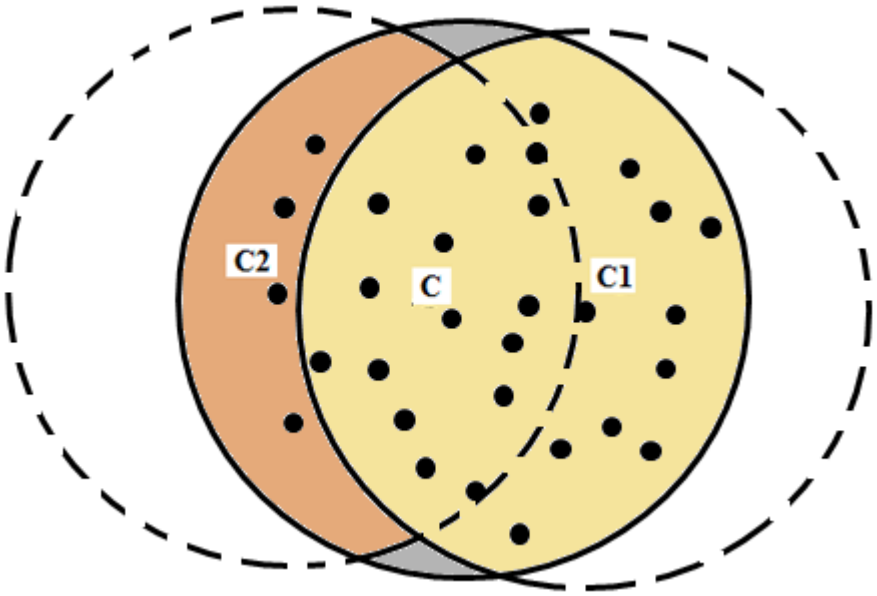

Figura 3.2: Exemplo do interior de um agrupamento da RLC.

Dado que um dos problemas identificados nesta estrutura está relacionado com o tamanho dos vectores de distâncias de cada ponto aos centros dos agrupamentos em que está contido, o principal objectivo da variante proposta é fazer com que a estrutura cresça mais em largura e menos em profundidade. Assim, os pontos dos agrupamentos mais profundos terão vectores de menor dimensão.

Assim surge a Recursive Lists of Clusters 2 (RLC2). A ideia desta variante consiste na redução progressiva do raio dos agrupamentos à medida que a profundidade destes aumenta. Desta forma, decidiu-se criar uma função que relacionasse o raio dos agrupamentos de nível zero com o nível de cada agrupamento para calcular o raio de qualquer agrupamento da estrutura. Deste modo, o raio de qualquer agrupamento da estrutura pode ser obtido a partir da expressão $r = r' / (n + 1)$, em que r' é o raio dos agrupamentos de nível zero e n é o nível do agrupamento. O raio da lista de agrupamentos de nível zero continua a ser um parâmetro desta estrutura, tal como na RLC.

A partir da figura 3.3 pode-se observar um exemplo do interior de um agrupamento da RLC2. O interior do agrupamento com centro C consiste numa lista com sete agrupamentos.

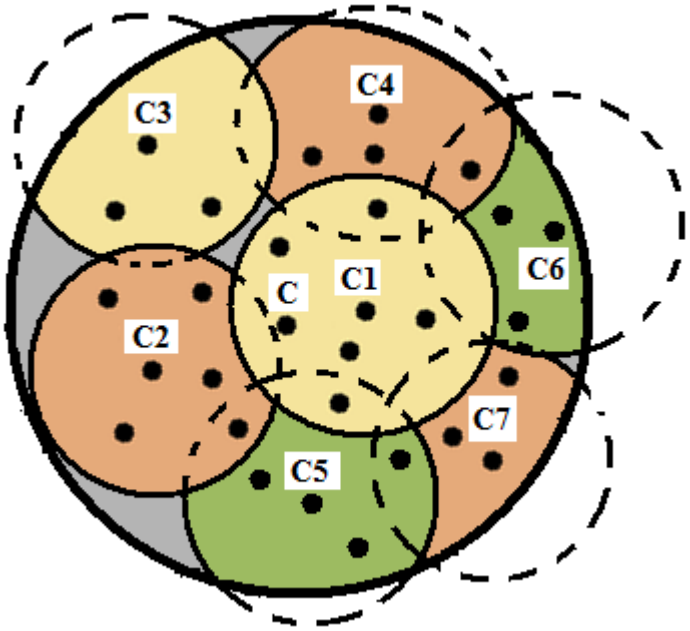

Figura 3.3: Exemplo do interior de um agrupamento da RLC2.

Em comparação com a RLC, quando uma folha da RLC2 atinge a sua capacidade máxima e é transformada numa lista de agrupamentos, os seus pontos serão melhor distribuídos pelos agrupamentos abaixo.

3.3 Definição Original da Recursive Lists of Clusters (RLC0)

A definição original da RLC [Mam05] definia o raio dos agrupamentos através de dois parâmetros: um real positivo ρ (que será o raio dos agrupamentos de nível zero) e uma função $\varphi : \mathbb{R}^+ \rightarrow \mathbb{R}^+$. O raio de um agrupamento de nível n é $\varphi^n(\rho)$, onde $\varphi^0(\rho) = \rho$ e $\varphi^n(\rho) = \varphi\ (\ (\varphi^{n-1}(\rho)\ )$, para qualquer n = 1, 2,…

Nos exemplos e nos testes experimentais apresentados em [Mam05], $\varphi\ (\rho)$ = k * ρ, onde k é uma constante positiva. Nestes casos, o raio de um agrupamento de nível n é k^n * ρ. Esta versão da RLC será denominada por Recursive Lists of Clusters 0 (RLC0).

Como não existia nenhuma implementação da RLC0 em memória secundária, foi implementada uma versão onde $\varphi\ (\rho)$ = k * ρ.

4. Espaços Métricos Utilizados

A selecção dos espaços métricos a usar nos testes baseou-se na escolha de dados reais. Como tal, foram escolhidos quatro domínios de características distintas: um dicionário de alemão, um dicionário de inglês, um conjunto de histogramas de imagens e um conjunto de características extraídas a partir de imagens de rosto. Em nenhum destes conjuntos existem repetições de elementos.

De seguida é apresentada a descrição de cada espaço métrico utilizado, bem como uma análise à distribuição das distâncias dos pontos de cada um dos domínios. Nesta análise está presente a *dimensão intrínseca* de cada espaço métrico. Esta dimensão foi calculada a partir de uma fórmula proposta em [CNB+01]: $\rho = \mu^2 / 2\sigma^2$, em que ρ corresponde à dimensão do espaço, μ à média das distâncias e σ ao desvio padrão.

4.1 Dicionário de Alemão

Este espaço métrico consiste num conjunto de palavras em alemão[2] utilizando a distância de *Levenshtein*. A distância entre duas palavras é o número mínimo de operações necessárias para transformar uma palavra noutra. As operações consideradas são a inserção de um carácter, a remoção de um carácter e a substituição de um carácter por outro. A definição formal da distância de *Levenshtein* é a seguinte. Dada uma palavra X, seja x(i) o seu i-ésimo carácter. Dados dois caracteres a e b, define-se: dif(a, b) = 0, se a = b; e dif(a, b) = 1, no caso contrário.

[2] Base de dados obtida em http://www.sisap.org/

A distância entre duas palavras X e Y, de comprimentos m e n, respectivamente, é o valor da função $\text{distância}_{X,Y}(m, n)$, onde

$$\text{distância}_{X,Y}(i,j) = \begin{cases} i, & \text{se } i \geq 0 \text{ e } j = 0; \\ j, & \text{se } i = 0 \text{ e } j > 0; \\ \min(\ \text{distância}_{X,Y}(i-1, j-1) + \text{dif}(x_i, y_j), & \\ \quad 1 + \text{distância}_{X,Y}(i, j-1), & \\ \quad 1 + \text{distância}_{X,Y}(i-1, j)\ ), & \text{se } i > 0 \text{ e } j > 0. \end{cases}$$

Por questões de eficiência, a distância entre as palavras foi implementada iterativamente, aplicando a técnica da programação dinâmica.

Existem 74.916 palavras no dicionário, com dimensões que variam entre um e trinta e três. A partir da figura 4.1 e da tabela 4.1, podem ser observadas a distribuição das distâncias entre os pontos deste domínio e algumas propriedades das mesmas.

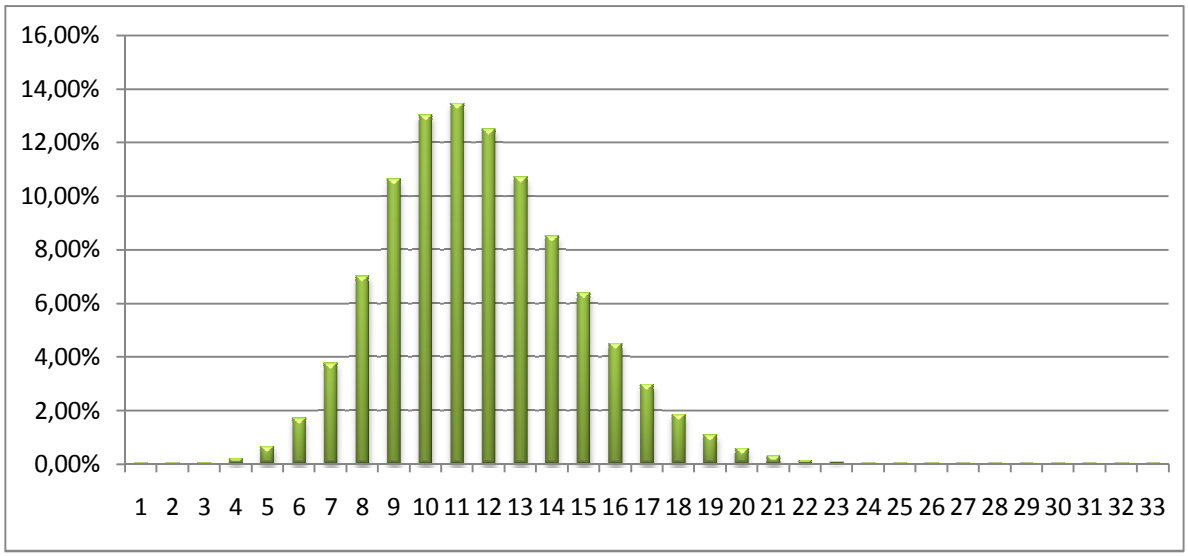

Figura 4.1: Distribuição das distâncias entre todas as palavras do dicionário de alemão, em relação ao número total de distâncias.

Número de distâncias	2.806.166.070
Distância mínima	1
Distância máxima	33
Média das distâncias	11,7
Variância	9,33
Desvio padrão	3,05
Dimensão do espaço	7,4

Tabela 4.1: Estatísticas das distâncias entre as palavras do dicionário de alemão.

4.2 Dicionário de Inglês

O dicionário de inglês é um conjunto de palavras em inglês que, tal como no caso anterior, foi obtido através da mesma fonte. Utiliza-se a mesma função de distância. A dimensão das 69.069 palavras que constituem este conjunto varia entre um e vinte e um.

Na tabela 4.2 encontram-se as estatísticas recolhidas a partir da análise das distâncias entre os pontos deste espaço métrico, enquanto que na figura 4.2 é possível observar a distribuição dessas mesmas distâncias.

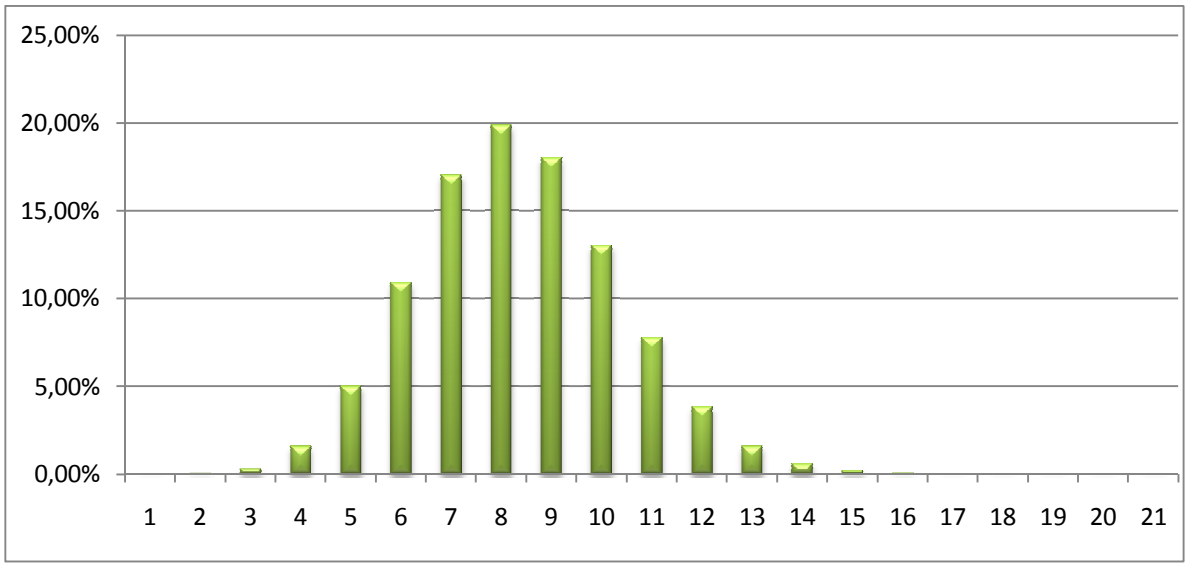

Figura 4.2: Distribuição das distâncias entre todas as palavras do dicionário de inglês, em relação ao número total de distâncias.

Número de distâncias	2.385.228.846
Distância mínima	1
Distância máxima	21
Média das distâncias	8,35
Variância	15,6
Desvio padrão	3,95
Dimensão do espaço	2,2

Tabela 4.2: Estatísticas das distâncias entre as palavras do dicionário de inglês.

4.3 Histogramas de Imagens

Este espaço métrico é composto por histogramas de imagens[3] em conjunto com a distância euclidiana. A distância euclidiana entre os pontos P = (p_1, p_2, …, p_n) e Q = (q_1, q_2, …, q_n) é dada pela fórmula $\sqrt{\sum_{i=1}^{n}(p_i - q_i)^2}$. Cada um dos 112.543 histogramas é uma sequência de cento e doze números reais.

Na tabela 4.3 encontram-se as estatísticas recolhidas a partir da análise das distâncias entre os pontos deste domínio. Os dados que estão presentes na figura 4.3 dizem respeito à distribuição deste conjunto de distâncias. No eixo horizontal estão presentes vinte e cinco intervalos de igual dimensão (que é aproximadamente igual a 0,0567). O primeiro intervalo corresponde às distâncias em [0,0001; 0,0568[, o segundo intervalo às distâncias em [0,0568; 0,1135[, e assim consecutivamente.

[3] Base de dados obtida em http://www.dbs.informatik.uni-muenchen.de/

Número de distâncias	6.332.907.153
Distância mínima	0,0001
Distância máxima	1,4171
Média das distâncias	0,43
Variância	0,03
Desvio padrão	0,17
Dimensão do espaço	3,1

Tabela 4.3: Estatísticas das distâncias entre os histogramas de imagens.

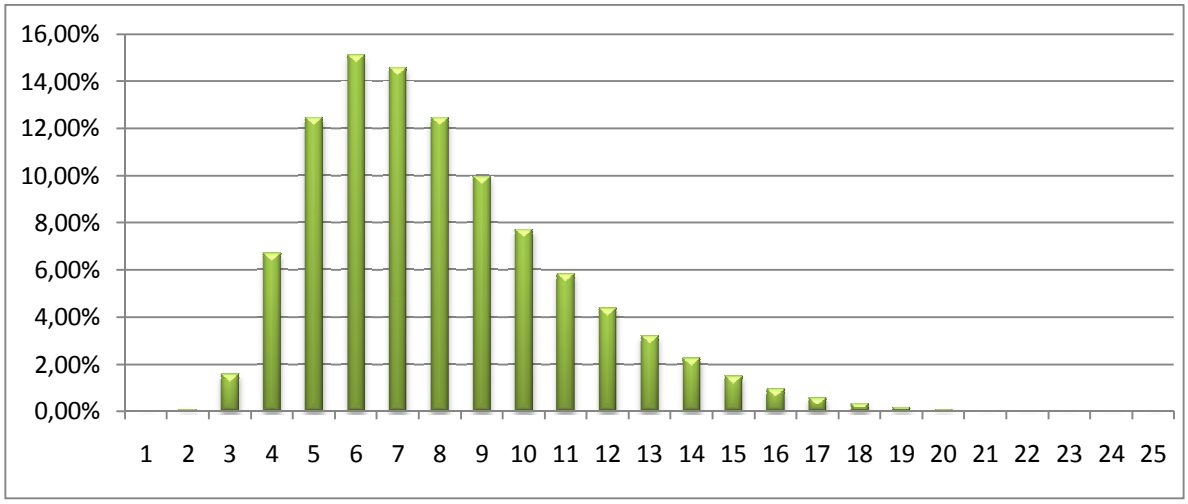

Figura 4.3: Distribuição das distâncias entre todos os histogramas de imagens, em relação ao número total de distâncias.

4.4 Imagens de Rosto

Este universo tem 3.040 vectores, cada um com vinte e quatro reais, extraídos de imagens de faces humanas. Usa-se a distância de Manhattan. A distância de *Manhattan* entre os pontos P = (p_1, p_2, …, p_n) e Q = (q_1, q_2, …, q_n) é obtida através da fórmula $\sum_{i=1}^{n}|p_i - q_i|$.

Os dados que fazem parte deste espaço métrico foram obtidos a partir do método *Eigenfaces* [TR91]. Neste método, cada imagem é representada por um vector de inteiros. O tamanho de cada vector corresponde ao número total de pixéis da imagem e cada elemento do vector corresponde ao valor de um pixel. A principal ideia deste método

consiste na redução da dimensão destes dados, de forma a se proceder ao reconhecimento facial num espaço de menores dimensões. Para tal, extraem-se as principais características a partir de um conjunto de treino. Estas características são extraídas usando um método matemático chamado PCA – *Principal Component Analysis*, obtendo-se um conjunto de vectores próprios. Intuitivamente, estes vectores conseguem representar a variação entre as imagens de rosto. Para cada imagem da base de dados, é então calculado um vector de pesos que corresponde à variação da imagem em relação aos vectores próprios do conjunto de treino. Assim, cada imagem da base de dados é representada por esse vector de pesos.

O conjunto de vectores de pesos que fazem parte da base de dados utilizada foram obtidos por Pedro Chambel, em [Cha09], que amavelmente os disponibilizou para a realização deste trabalho.

Na tabela 4.4 encontram-se as estatísticas referentes às distâncias entre os pontos constituintes deste espaço métrico, enquanto que na figura 4.4 se encontra a distribuição dessas mesmas distâncias em vinte e cinco intervalos de igual dimensão. Desta feita, no primeiro intervalo estão presentes as distâncias em [161; 2.564[, no segundo intervalo entre [2.564 e 4.967[, e assim consecutivamente.

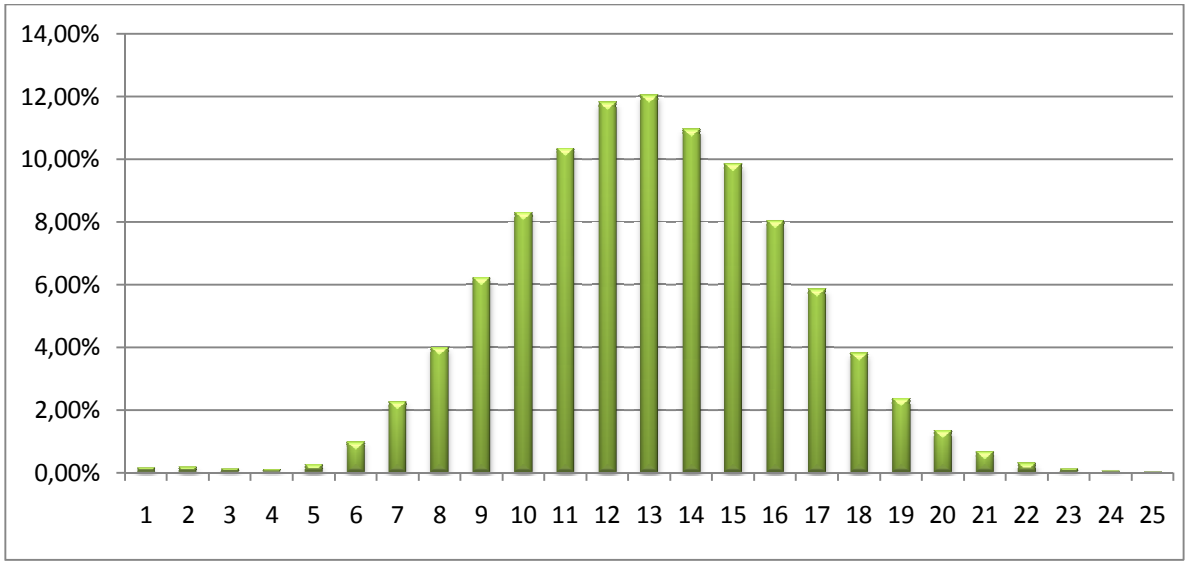

Figura 4.4: Distribuição das distâncias entre todas as imagens de rosto, em relação ao número total de distâncias.

Número de distâncias	4.619.280
Distância mínima	161
Distância máxima	60.231
Média das distâncias	30.203
Variância	61.892.745
Desvio padrão	7.867
Dimensão do espaço	7,4

Tabela 4.4: Estatísticas das distâncias entre as imagens de rosto.

5. Resultados Experimentais

As estruturas de dados escolhidas para os testes experimentais, para além da variante proposta, foram algumas daquelas sobre as quais o trabalho relacionado incidiu (secção 2.2).

As implementações da M-tree, Slim-tree e DF-tree fazem parte de uma biblioteca chamada *arboretum*[4], que foi desenvolvida pelo Grupo de Bases de Dados e Imagens do Instituto de Ciências Matemáticas e de Computação da Universidade de São Paulo. Houve uma tentativa de contacto com os autores da SM-tree e da D-Index, com o objectivo de obter uma implementação dessas mesmas estruturas, mas não foi recebida qualquer resposta.

Sendo assim, e como apenas a SM-tree permitiria remoções de objectos, não foi possível comparar resultados de remoções com nenhuma outra estrutura. Consequentemente, só foram realizados testes com remoções na RLC0, na RLC e na RLC2.

Antes da realização dos testes experimentais, foi necessário parametrizar cada uma das estruturas disponíveis. Seguidamente apresentam-se os parâmetros usados em cada uma delas (secção 5.1), bem como uma análise dos resultados obtidos (secção 5.2).

[4] Obtida em http://www.gbdi.icmc.usp.br/old/arboretum/

5.1 Parametrização das Estruturas de Dados

Para a parametrização das estruturas de dados que pertencem à biblioteca *arboretum*, foi consultada a bibliografia correspondente a cada estrutura para descobrir quais os melhores parâmetros. Os parâmetros da RLC e da RLC2 foram escolhidos através da comparação de resultados experimentais.

Nos testes realizados são usadas as seguintes métricas: número médio de cálculos de distâncias entre dois pontos durante inserções, número médio de cálculos de distâncias entre dois pontos durante pesquisas, número médio de leituras durante inserções, número médio de leituras durante pesquisas e número médio de escritas durante inserções.

A dimensão das páginas é um parâmetro comum a todas as estruturas de dados utilizadas. Essa dimensão varia com o espaço métrico testado. Dado que os pontos dos níveis muito profundos da RLC atingem dimensões bastante grandes, foram definidos os seguintes tamanhos para as páginas de todas as estruturas:

- 4.096 bytes para o espaço métrico das imagens de rosto. Como este espaço possui a menor de todas as bases de dados, é possível utilizar esta dimensão.
- 8.192 bytes para os dicionários de palavras. Devido à limitação da implementação da RLC, tiveram que ser usadas páginas desta dimensão.
- 16.384 bytes para os histogramas de imagens. Uma vez que este espaço métrico possui a maior de todas as bases de dados, não foi possível realizar testes com páginas de menor dimensão.

Em anexo, apresentam-se resultados experimentais que comparam o desempenho da M-tree, da Slim-tree, da DF-tree e da RLC2, usando páginas de 4.096 bytes.

5.1.1 Parâmetros da M-tree

No que diz respeito à M-tree, é possível escolher a forma segundo a qual se realiza o particionamento dos seus nós e o método de promoção a utilizar. Segundo [CPZ97], deve

ser usado o método do hiperplano generalizado para particionar os nós e o método *m_RAD* como método de promoção.

5.1.2 Parâmetros da Slim-tree

Esta estrutura permite escolher a sub-árvore onde se insere um objecto quando mais do que uma está em condições de o receber, o método de particionamento dos nós e a possibilidade de usar o algoritmo *Slim-down*. Para os primeiros dois parâmetros, foram escolhidos, respectivamente, de acordo com Traina et al. em [TTF+02], os métodos *minoccup* e árvore mínima de cobertura.

Apesar de o método *Slim-down* ter sido identificado como benéfico para o desempenho da Slim-tree, a sua implementação não constava na biblioteca *arboretum*, razão pela qual não foi utilizado.

5.1.3 Parâmetros da DF-tree

Para além dos mesmos parâmetros que a Slim-tree, esta estrutura apresenta mais dois parâmetros: o número de representantes globais e o limite do algoritmo *when to update*.

Em relação aos parâmetros comuns, as escolhas foram as mesmas.

Em [TTS+02] consta que o número de representantes globais deve corresponder à dimensionalidade intrínseca do espaço métrico mas, depois de experimentados esses e outros valores, verificou-se que o desempenho da estrutura, no que diz respeito às métricas relacionadas com as inserções, melhorava quanto maior fosse o valor utilizado. Os resultados relacionados com as pesquisas foram sempre os mesmos. Como os valores das dimensões intrínsecas dos espaços métricos utilizados rondam as poucas unidades, foram usados os valores 33, 21, 112 e 24 para os testes com o dicionário de alemão, o dicionário de inglês, os histogramas de imagens e as imagens de rosto, respectivamente. Estes valores correspondem à dimensão máxima dos pontos destes domínios.

No que diz respeito ao limite do algoritmo *when to update*, existe, em [TTS+02], um teste experimental em que é utilizado o valor 2.000 para este parâmetro. Tanto este valor como uma vasta gama de valores menores e maiores foram experimentados, mas os resultados obtidos nunca se alteraram. Desta forma, foi escolhido o valor 2.000 para todos os testes realizados.

5.1.4 Parâmetros da Recursive Lists of Clusters

A RLC dispõe de dois parâmetros, o raio dos agrupamentos e a capacidade das folhas, que foram escolhidos pela observação de resultados experimentais. Os testes utilizados para esta parametrização consistem na inserção de todos os objectos de cada espaço métrico e de um número de pesquisas por proximidade que ronda os 10% do número total de objectos inseridos. Os objectos de pesquisa foram escolhidos aleatoriamente a partir dos pontos do espaço métrico e os raios de pesquisa foram gerados aleatoriamente, dentro de um determinado intervalo.

Para cada teste começou-se por variar o raio dos agrupamentos, mantendo a capacidade das folhas fixa. De seguida, com o melhor raio já encontrado, variou-se a capacidade das folhas mantendo o raio fixo.

Um dos pormenores da implementação que foi utilizada está relacionado com uma fórmula que calcula o raio dos agrupamentos com base em alguns valores que caracterizam o domínio dos espaços métricos euclidianos testados em [Rod06]. Essa fórmula foi utilizada e, por isso, os valores testados para os raios dos agrupamentos podem ser pouco comuns.

Será possível observar que o valor do raio dos agrupamentos influencia bastante mais os resultados do que a capacidade das folhas. Apesar das diferenças entre as capacidades das folhas, alguns resultados obtidos eram praticamente equivalentes, tendo obrigatoriamente que se eleger um deles como óptimo.

Seguidamente são apresentados os melhores parâmetros encontrados para cada espaço métrico.

Dicionário de Alemão

Raio dos agrupamentos	Capacidade das folhas	Nº médio de cálculos de distâncias entre dois pontos durante inserções	Nº médio de cálculos de distâncias entre dois pontos durante pesquisas	Nº médio de leituras durante inserções	Nº médio de leituras durante pesquisas	Nº médio de escritas durante inserções
1	200	33.549	69.400	200	416	1,1
2	200	24.963	57.500	1.491	347	0,1
3	200	15.874	43.564	102	276	8,2
4	100	8.777	30.613	83	274	32
4	200	8.776	30.654	119	257	68
4	400	8.774	30.686	140	233	89

Tabela 5.1: Testes realizados para a parametrização da RLC com o dicionário de alemão.

A partir da tabela 5.1, foram escolhidos os valores 4 e 100 para o raio dos agrupamentos e a capacidade das folhas, respectivamente. Estes valores foram escolhidos porque minimizam o número médio de cálculos de distâncias entre dois pontos durante pesquisas e o número médio de leituras durante inserções. Apesar de existir uma tendência para que se obtenham melhores resultados quanto maior for o raio dos agrupamentos, não foi possível continuar a aumentar os mesmos. A dimensão dos pontos do último nível da estrutura já excedia uma página e, de acordo com a implementação da RLC, tal não pode acontecer.

Dicionário de Inglês

Tendo por base os resultados da tabela 5.2, foi escolhido o valor 3 para o raio dos agrupamentos e o valor 1.500 para a capacidade das folhas. A escolha destes valores tem por base os resultados das operações de acesso a disco. Tal como no caso do dicionário de

alemão, não foi possível testar agrupamentos com raio maior por causa das dimensões atingidas pelos pontos do último nível da estrutura.

Raio dos agrupamentos	Capacidade das folhas	Nº médio de cálculos de distâncias entre dois pontos durante inserções	Nº médio de cálculos de distâncias entre dois pontos durante pesquisas	Nº médio de leituras durante inserções	Nº médio de leituras durante pesquisas	Nº médio de escritas durante inserções
1	75	16.979	36.442	76	238	1,4
2	75	8.492	21.838	38	235	1,7
3	50	3.439	13.714	25	640	11
3	75	3.438	13.806	29	554	15
3	150	3.437	13.945	34	392	20
3	300	3.434	14.133	28	266	14
3	600	3.433	14.373	16	208	1,7
3	1.500	3.433	14.485	16	203	1,7

Tabela 5.2: Testes realizados para a parametrização da RLC com o dicionário de inglês.

Histogramas de Imagens

O valor escolhido para o raio dos agrupamentos foi 0,1825 enquanto que, para a capacidade das folhas, se escolheu o valor 25. Neste caso foi necessário fazer um compromisso pois, apesar de se poderem usar raios maiores (e menores), foram estes os valores que minimizaram o número médio de escritas durante inserções, valor este que continua bastante elevado, tal como apresentado na tabela 5.3.

Raio dos agrupamentos	Capacidade das folhas	Nº médio de cálculos de distâncias entre dois pontos durante inserções	Nº médio de cálculos de distâncias entre dois pontos durante pesquisas	Nº médio de leituras durante inserções	Nº médio de leituras durante pesquisas	Nº médio de escritas durante inserções
0,1890	100	654	9.172	246	1.732	213
0,1857	100	733	9.571	228	1.654	190
0,1825	25	835	10.183	109	2.432	64
0,1825	50	832	10.057	134	1.987	89
0,1825	100	829	10.102	228	1.604	183
0,1825	150	818	10.172	268	1.409	224

Tabela 5.3: Testes realizados para a parametrização da RLC com os histogramas de imagens.

Imagens de Rosto

Raio dos agrupamentos	Capacidade das folhas	Nº médio de cálculos de distâncias entre dois pontos durante inserções	Nº médio de cálculos de distâncias entre dois pontos durante pesquisas	Nº médio de leituras durante inserções	Nº médio de leituras durante pesquisas	Nº médio de escritas durante inserções
18.528	50	24	510	12	125	12
15.440	50	46	387	4,3	63	3,2
15.440	70	46	417	3,9	56	2,7
15.440	90	45	442	3,6	51	2,5
13.234	50	68	308	4,2	46	2,0
11.580	50	84	314	4,8	39	1,8

Tabela 5.4: Testes realizados para a parametrização da RLC com as imagens de rosto.

Como este ficheiro de dados era, de longe, o de menores dimensões, foi possível variar o raio dos agrupamentos até encontrar o valor mais apropriado. Assim, foi escolhido o valor 15.440 para esse parâmetro. O valor 90 foi o escolhido para a capacidade das

folhas. Estes valores foram escolhidos porque minimizam o número médio de leituras durante inserções.

Os resultados dos testes para a parametrização desta estrutura com este espaço métrico podem ser consultados na tabela 5.4.

5.1.5 Parâmetros da Recursive Lists of Clusters 2

O método para parametrizar a RLC2 foi o mesmo que o utilizado na RLC, tendo-se efectuado as mesmas operações, exactamente pela mesma ordem.

Tal como na RLC, a capacidade das folhas não influencia tanto o desempenho da estrutura como o raio dos agrupamentos. Na RLC2, a dificuldade da escolha do melhor valor para a capacidade das folhas ainda foi maior, pois, para o mesmo raio, as diferenças dos resultados são mínimas. A seguir apresentam-se os parâmetros encontrados para cada espaço métrico.

Dicionário de Alemão

A partir dos resultados que constam na tabela 5.5, foi escolhido o valor 10 para o raio dos agrupamentos e o valor 100 para a capacidade das folhas.

Raio dos agrupamentos	Capacidade das folhas	Nº médio de cálculos de distâncias entre dois pontos durante inserções	Nº médio de cálculos de distâncias entre dois pontos durante pesquisas	Nº médio de leituras durante inserções	Nº médio de leituras durante pesquisas	Nº médio de escritas durante inserções
9	100	541	10.109	4,4	201	2,1
10	50	356	8.968	3,5	243	2,3
10	100	357	9.034	3,4	232	2,2
10	200	358	9.217	3,4	217	2,2
11	100	511	10.149	4,6	231	2,3

Tabela 5.5: Testes realizados para a parametrização da RLC2 com o dicionário de alemão.

Dicionário de Inglês

Tal como no dicionário de alemão e a partir da tabela 5.6, foi escolhido o valor 100 para a capacidade das folhas. Para o raio dos agrupamentos foi escolhido o valor 6.

Raio dos agrupamentos	Capacidade das folhas	Nº médio de cálculos de distâncias entre dois pontos durante inserções	Nº médio de cálculos de distâncias entre dois pontos durante pesquisas	Nº médio de leituras durante inserções	Nº médio de leituras durante pesquisas	Nº médio de escritas durante inserções
5	100	331	11.197	2,9	359	2,2
6	50	247	9.895	2,4	481	2,1
6	100	247	10.279	2,3	412	2,0
6	200	247	10.769	2,2	367	1,9
7	100	543	11.050	3,2	309	1,8

Tabela 5.6: Testes realizados para a parametrização da RLC2 com o dicionário de inglês.

Histogramas de Imagens

Raio dos agrupamentos	Capacidade das folhas	Nº médio de cálculos de distâncias entre dois pontos durante inserções	Nº médio de cálculos de distâncias entre dois pontos durante pesquisas	Nº médio de leituras durante inserções	Nº médio de leituras durante pesquisas	Nº médio de escritas durante inserções
1,06	50	21	3.349	7,0	675	7,1
0,71	25	42	4.258	5,8	753	5,3
0,71	50	36	4.076	5,7	685	5,4
0,71	100	34	3.862	6,4	604	6,1
0,53	50	66	4.940	6,3	690	4,6
0,34	20	305	7.426	77	3.171	4,9

Tabela 5.7: Testes realizados para a parametrização da RLC2 com os histogramas de imagens.

De acordo com a informação disponível na tabela 5.7, foram escolhidos os valores 0,71 e 50 para o raio dos agrupamentos e para a capacidade das folhas, respectivamente.

Imagens de Rosto

Tendo em conta a tabela 5.8, foi escolhido o valor 23.160 para o raio dos agrupamentos, enquanto que 25 foi o valor escolhido para a capacidade das folhas.

Raio dos agrupamentos	Capacidade das folhas	Nº médio de cálculos de distâncias entre dois pontos durante inserções	Nº médio de cálculos de distâncias entre dois pontos durante pesquisas	Nº médio de leituras durante inserções	Nº médio de leituras durante pesquisas	Nº médio de escritas durante inserções
30.880	50	19	409	2,7	73	2,7
23.160	25	19	297	2,2	62	2,2
23.160	50	20	316	2,3	60	2,3
23.160	100	20	403	2,7	61	2,8
18.528	50	25	341	2,3	55	2,2
15.440	50	47	336	3,2	49	2,1

Tabela 5.8: Testes realizados para a parametrização da RLC2 com as imagens de rosto.

5.2 Análise Empírica das Estruturas de Dados

Os resultados que serão seguidamente apresentados resultam de uma média aritmética dos resultados de um conjunto de três testes distintos com cada espaço métrico. Cada teste de cada domínio consiste exactamente nas mesmas inserções e nas mesmas pesquisas, apenas a ordem das inserções varia de teste para teste. A ordem das inserções é diferente pois a forma das estruturas varia conforme a ordem pela qual os objectos são inseridos.

Em cada teste é inserida a totalidade dos pontos do respectivo espaço métrico e são realizadas 7.500, 6.900, 11.300 e 300 pesquisas por proximidade para os testes do dicionário de alemão, do dicionário de inglês, dos histogramas de imagens e das imagens de rosto, respectivamente. Estes valores correspondem a cerca de 10% do número de objectos do domínio. Os parâmetros usados em cada estrutura de dados foram indicados na secção 5.1.

5.2.1 Comparação de Resultados Experimentais

Dicionário de Alemão

Como pode ser observado na tabela 5.9, a Slim-tree obtém melhores resultados no número médio de cálculos de distâncias entre dois pontos durante inserções e no número médio de leituras durante inserções. Em todas as outras métricas, a RLC2 é a estrutura que regista o melhor desempenho. Numa comparação restrita à RLC e RLC2, pode-se verificar que houve grandes melhorias em cada uma das métricas.

	M-tree	Slim-tree	DF-tree	RLC	RLC2
Nº médio de cálculos de distâncias entre dois pontos durante inserções	32.610	143	2.788	8.744	371
Nº médio de cálculos de distâncias entre dois pontos durante pesquisas	24.512	48.555	22.713	22.486	8.585
Nº médio de leituras durante inserções	2,9	2,8	16	89	3,5
Nº médio de leituras durante pesquisas	554	483	554	332	214
Nº médio de escritas durante inserções	3,9	3,8	4,9	38	2,3

Tabela 5.9: Resultados dos testes com o dicionário de alemão.

Dicionário de Inglês

A partir da tabela 5.10 é possível observar-se que a RLC2 volta a ter um lugar de destaque, obtendo os melhores resultados no número médio de cálculos de distâncias entre dois pontos durante pesquisas e no número médio de leituras durante inserções. É de assinalar que registou o pior desempenho, entre todas as estruturas, no que diz respeito ao número médio de leituras durante pesquisas. A Slim-tree voltou a ter os melhores resultados na métrica relativa ao número médio de cálculos de distâncias entre dois pontos durante inserções.

Na comparação directa entre a RLC e a RLC2, existiram melhorias no número médio de cálculos de distâncias entre dois pontos durante inserções, no número médio de cálculos de distâncias entre dois pontos durante pesquisas e no número médio de leituras durante inserções. Em oposição, nas restantes métricas, o desempenho da RLC2 foi inferior ao da RLC.

	M-tree	Slim-tree	DF-tree	RLC	RLC2
Nº médio de cálculos de distâncias entre dois pontos durante inserções	61.485	195	3.736	3.450	250
Nº médio de cálculos de distâncias entre dois pontos durante pesquisas	26.037	49.108	32.522	14.610	10.533
Nº médio de leituras durante inserções	2,7	2,6	17	16	2,3
Nº médio de leituras durante pesquisas	397	333	367	205	409
Nº médio de escritas durante inserções	3,7	3,6	4,7	1,7	2,0

Tabela 5.10: Resultados dos testes com o dicionário de inglês.

Histogramas de Imagens

A partir da observação da tabela 5.11 regista-se que, de todas as métricas, a RLC2 só não obtém o melhor resultado no número médio de leituras durante inserções. Volta também a registar-se uma melhoria, em todos os aspectos, da RLC2 em relação à RLC.

	M-tree	Slim-tree	DF-tree	RLC	RLC2
Nº médio de cálculos de distâncias entre dois pontos durante inserções	578	54	1.073	836	37
Nº médio de cálculos de distâncias entre dois pontos durante pesquisas	18.646	32.444	16.922	10.165	3.974
Nº médio de leituras durante inserções	4,8	4,8	79	109	5,7
Nº médio de leituras durante pesquisas	3.288	5.373	5.336	2.431	676
Nº médio de escritas durante inserções	6	6	6,9	65	5,4

Tabela 5.11: Resultados dos testes com os histogramas de imagens.

Imagens de Rosto

De acordo com a tabela 5.12, a superioridade da RLC2 volta a ser demonstrada em todas as medidas de comparação, com excepção do número médio de leituras durante pesquisas. Foi nessa métrica que a RLC obteve o melhor resultado, sendo novamente superada pela RLC2 em todas as outras métricas.

	M-tree	Slim-tree	DF-tree	RLC	RLC2
Nº médio de cálculos de distâncias entre dois pontos durante inserções	625	38	166	44	20
Nº médio de cálculos de distâncias entre dois pontos durante pesquisas	1.131	1.869	1.015	448	309
Nº médio de leituras durante inserções	3,1	3	10	3,5	2,2
Nº médio de leituras durante pesquisas	150	176	181	51	62
Nº médio de escritas durante inserções	4,2	4	5,1	2,4	2,2

Tabela 5.12: Resultados dos testes com as imagens de rosto.

5.2.2 Recursive Lists of Clusters 2 *versus* Recursive Lists of Clusters 0

Para a comparação entre a RLC2 e a RLC0, foi usado apenas um ficheiro de teste para cada espaço métrico.

Os raios dos agrupamentos da RLC0, tal como nos testes experimentais em [Mam05], são obtidos através da função $\varphi(\rho) = k * \rho$. Como, para estes testes, foi escolhido o valor 0,5 para k, o raio de um agrupamento nível n da RLC0 de será $0{,}5^n * \rho$.

Uma vez que, em ambas as estruturas, os raios dos agrupamentos vão diminuindo à medida que a profundidade aumenta, foi possível executar todos os testes usando páginas com 4096 bytes. Os parâmetros usados nas duas estruturas também foram os mesmos pois, depois de alguns testes, foi verificado que aqueles que foram considerados os melhores parâmetros para a RLC2 também o são para a RLC0.

Nas tabelas 5.13, 5.14, 5.15 e 5.16 estão presentes os resultados dos testes com o dicionário de alemão, o dicionário de inglês, os histogramas de imagens e as imagens de rosto, respectivamente.

	RLC2	RLC0
Nº médio de cálculos de distâncias entre dois pontos durante inserções	357	368
Nº médio de cálculos de distâncias entre dois pontos durante pesquisas	9.034	9.297
Nº médio de leituras durante inserções	5,8	5,9
Nº médio de leituras durante pesquisas	288	271
Nº médio de escritas durante inserções	2,4	2,5

Tabela 5.13: Resultados dos testes da RLC2 e RLC0 com o dicionário de alemão.

	RLC2	RLC0
Nº médio de cálculos de distâncias entre dois pontos durante inserções	247	254
Nº médio de cálculos de distâncias entre dois pontos durante pesquisas	10.279	10.430
Nº médio de leituras durante inserções	3,4	3,4
Nº médio de leituras durante pesquisas	449	425
Nº médio de escritas durante inserções	2,2	2,2

Tabela 5.14: Resultados dos testes da RLC2 e RLC0 com o dicionário de inglês.

	RLC2	RLC0
Nº médio de cálculos de distâncias entre dois pontos durante inserções	36,4	306
Nº médio de cálculos de distâncias entre dois pontos durante pesquisas	4.076	8.270
Nº médio de leituras durante inserções	14,4	78,9
Nº médio de leituras durante pesquisas	1.498	2.226
Nº médio de escritas durante inserções	9,62	6,51

Tabela 5.15: Resultados dos testes da RLC2 e RLC0 com os histogramas de imagens.

	RLC2	RLC0
Nº médio de cálculos de distâncias entre dois pontos durante inserções	19	19
Nº médio de cálculos de distâncias entre dois pontos durante pesquisas	297	296
Nº médio de leituras durante inserções	2,2	2,1
Nº médio de leituras durante pesquisas	62	61
Nº médio de escritas durante inserções	2,2	2,2

Tabela 5.16: Resultados dos testes da RLC2 e RLC0 com as imagens de rosto.

Como é possível observar através das quatro tabelas anteriores, a RLC2 continua a mostrar alguma superioridade em relação à "concorrência".

Exceptuando o teste das imagens de rosto, no qual as diferenças dos resultados são desprezáveis, a RLC2 revela melhores desempenhos na maioria das métricas testadas. Desta forma é possível afirmar que, apesar da RLC0 apresentar resultados muito bons, a RLC2 apresenta resultados melhores.

5.2.3 Remoções na RLC0, na RLC e na RLC2

Uma vez que não se conseguiu obter nenhuma estrutura de dados cujo algoritmo de remoção estivesse implementado, decidiu-se comparar o desempenho da operação de remoção nas únicas estruturas disponíveis que permitem a execução dessa operação. Para tal, foram utilizados os mesmos espaços métricos que anteriormente e foram criados três novos ficheiros de teste para cada espaço métrico. Os resultados que serão apresentados resultam da média aritmética dos resultados desses três testes. Aos ficheiros de testes antigos, foi alterada a ordem de inserção dos objectos e a quantidade de pesquisas que existiam nos testes anteriores foi substituída pelo mesmo número remoções, ou seja, cerca de 10% do número de inserções.

Os parâmetros usados em cada estrutura não sofreram qualquer alteração.

	RLC0	RLC	RLC2
Nº médio de cálculos de distâncias entre dois pontos durante remoções	2.371	17.368	2.307
Nº médio de leituras durante remoções	20	7.025	20
Nº médio de escritas durante remoções	9,4	6.923	9,4

Tabela 5.17: Remoções na RLC0, RLC e RLC2 com o dicionário de alemão.

	RLC0	RLC	RLC2
Nº médio de cálculos de distâncias entre dois pontos durante remoções	1.242	8.732	1.222
Nº médio de leituras durante remoções	9,6	43	9,7
Nº médio de escritas durante remoções	6,5	3,4	6,6

Tabela 5.18: Remoções na RLC0, RLC e RLC2 com o dicionário de inglês.

	RLC0	RLC	RLC2
Nº médio de cálculos de distâncias entre dois pontos durante remoções	1.273	12.938	228
Nº médio de leituras durante remoções	83	15.435	27
Nº médio de escritas durante remoções	13	15.292	21

Tabela 5.19: Remoções na RLC0, RLC e RLC2 com os histogramas de imagens.

	RLC0	RLC	RLC2
Nº médio de cálculos de distâncias entre dois pontos durante remoções	93	194	93
Nº médio de leituras durante remoções	8,6	15	8,6
Nº médio de escritas durante remoções	7,2	8,0	7,5

Tabela 5.20: Remoções na RLC0, RLC e RLC2 com as imagens de rosto.

A partir das tabelas 5.17, 5.18, 5.19 e 5.20 verifica-se, mais uma vez, um grande equilíbrio entre a RLC0 e a RLC2, sendo estas, sem qualquer dúvida, as estruturas que apresentam melhor desempenho. Mesmo assim, considerando os valores que apresentam maiores diferenças e que, portanto, são menos desprezáveis, a RLC2 revela uma certa ascendência sobre a RLC0.

5.2.4 Considerações sobre a Recursive Lists of Clusters 2

Como previsto, a redução dos raios dos agrupamentos à medida que a profundidade aumenta, causou alterações à forma final da RLC2 em comparação com a da RLC. Para uma melhor compreensão dessas diferenças, realizaram-se alguns testes utilizando apenas um ficheiro de cada espaço métrico. Os resultados desses testes podem ser observados nas tabelas 5.21 e 5.22.

Espaço métrico	Nº total de agrupamentos	Nº máximo de pontos num agrupamento	Comprimento da primeira lista de agrupamentos	Comprimento máximo de outra lista de agrupamentos	Profundidade máxima
Dicionário de alemão	32.186	1.339	30.394	20	670
Dicionário de inglês[5]	14.486	1.106	14.486	0	1
Histogramas de imagens	44.980	6.105	9.890	52	1.512
Imagens de rosto	167	192	142	9	4

Tabela 5.21: Forma final da RLC para cada espaço métrico.

Espaço métrico	Nº total de agrupamentos	Nº máximo de pontos num agrupamento	Comprimento da primeira lista de agrupamentos	Comprimento máximo de outra lista de agrupamentos	Profundidade máxima
Dicionário de alemão	22.976	18.169	2.360	2.494	4
Dicionário de inglês	18.559	19.696	2.281	1.033	4
Histogramas de imagens	25.747	104.010	91	422	9
Imagens de rosto	326	628	42	67	2

Tabela 5.22: Forma final da RLC2 para cada espaço métrico.

Uma das conclusões que pode ser imediatamente extraída está relacionada com o equilíbrio de cada estrutura. Na RLC2, o comprimento da primeira lista de agrupamentos está muito mais próximo do maior comprimento de uma lista de agrupamentos de nível

[5] A profundidade da RLC neste teste pode ser considerada excepcional pois o valor que foi escolhido para a capacidade das folhas (1.500) é superior ao número máximo de pontos num agrupamento (secção 5.1.4).

superior. Este facto torna a RLC2 mais equilibrada que a RLC, onde a primeira lista de agrupamentos é muito mais longa que as restantes.

No que diz respeito ao número máximo de pontos num agrupamento, é lógico que este tenha aumentado. Uma vez que o raio dos agrupamentos da primeira lista é substancialmente maior do que o utilizado na RLC, mais pontos ficam contidos em cada agrupamento. O facto dos agrupamentos de nível zero da RLC2 serem maiores que os da RLC provoca também que a primeira lista de agrupamentos da RLC2 seja menor que a da RLC.

Dado que a RLC2 é, geralmente, menos profunda que a RLC, os pontos dos últimos níveis da RLC2 têm associados a si vectores de distâncias aos centros dos agrupamentos em que estão contidos bastante menores que os da RLC. Desta forma, esses pontos irão ocupar menos espaço em disco. Este facto contribui significativamente para a redução do número de acessos a disco. Uma característica das implementações destas duas estruturas é que, sempre que é feita uma leitura ou uma escrita, o número de bytes lidos ou escritos é o equivalente a uma página. Uma vez que os pontos mais profundos da RLC2 ocupam menos espaço, sempre que se lê ou se escreve, por exemplo, uma página de uma folha, são lidos ou escritos mais pontos do que anteriormente.

Como as melhorias da RLC2 não foram apenas em operações de acesso a disco, é necessário que se compreenda por que razão diminuiu o número de distâncias calculadas, quer nas pesquisas quer nas inserções. Para se entender como é que se obtiveram tais ganhos de desempenho, é necessário consultar as tabelas 5.23 e 5.24. Nessas tabelas pode-se constatar que houve um decréscimo acentuado do número médio de agrupamentos visitados, tanto nas inserções como nas pesquisas, da RLC para a RLC2. Por cada agrupamento a menos que é visitado, poupa-se, pelo menos, o cálculo da distância ao centro desse agrupamento.

Espaço métrico	Nº médio de agrupamentos visitados por inserção	Nº médio de agrupamentos visitados por pesquisa
Dicionário de alemão	2.611	826.535
Dicionário de inglês	1.914	1.049.256
Histogramas de imagens	400	506.919
Imagens de rosto	42	1.973

Tabela 5.23: Número médio de agrupamentos visitados, por espaço métrico, na RLC.

Espaço métrico	Nº médio de agrupamentos visitados por inserção	Nº médio de agrupamentos visitados por pesquisa
Dicionário de alemão	207	143.610
Dicionário de inglês	168	194.857
Histogramas de imagens	29	30.281
Imagens de rosto	17	1.087

Tabela 5.24: Número médio de agrupamentos visitados, por espaço métrico, na RLC2.

Forma da RLC2

À medida que se foram variando os parâmetros da RLC2, para um único teste de cada domínio, verificou-se que, à excepção dos histogramas de imagens, quanto maior for a proximidade entre a dimensão da primeira lista de agrupamentos e a dimensão da maior lista de agrupamentos de nível superior, melhor é o desempenho (vejam-se as tabelas 5.25, 5.26, 5.27 e 5.28). No caso das imagens de rosto, os parâmetros escolhidos não maximizam essa proximidade, mas a diferença entre o valor obtido e o máximo é insignificante.

Para além da relação entre a dimensão das listas de agrupamentos, verifica-se também que, tipicamente, a RLC2 atinge poucas unidades de profundidade.

Raio dos agrupamentos	Capacidade das folhas	Comprimento da primeira lista de agrupamentos	Comprimento máximo de outra lista de agrupamentos	Valor absoluto da diferença entre os comprimentos das listas	Profundidade máxima
9	100	3.621	3.100	521	4
10	50	2.360	2.494	134	4
10	100	2.360	2.494	134	4
10	200	2.360	2.505	145	3
11	100	1.546	3.870	2.324	4

Tabela 5.25: Forma final da RLC2 com o dicionário de alemão, usando diferentes parâmetros.

Raio dos agrupamentos	Capacidade das folhas	Comprimento da primeira lista de agrupamentos	Comprimento máximo de outra lista de agrupamentos	Valor absoluto da diferença entre os comprimentos das listas	Profundidade máxima
5	100	4.338	1.391	2.947	3
6	50	2.281	1.032	1.249	4
6	100	2.281	1.033	1.248	4
6	200	2.281	1.033	1.248	3
7	100	1.213	2.499	1.286	4

Tabela 5.26: Forma final da RLC2 com o dicionário de inglês, usando diferentes parâmetros.

Raio dos agrupamentos	Capacidade das folhas	Comprimento da primeira lista de agrupamentos	Comprimento máximo de outra lista de agrupamentos	Valor absoluto da diferença entre os comprimentos das listas	Profundidade máxima
1,06	50	36	125	89	14
0,71	25	91	552	461	12
0,71	50	91	422	331	9
0,71	100	91	432	341	8
0,53	50	150	901	751	8
0,34	20	600	4.470	3.870	7

Tabela 5.27: Forma final da RLC2 com os histogramas de imagens, usando diferentes parâmetros.

Raio dos agrupamentos	Capacidade das folhas	Comprimento da primeira lista de agrupamentos	Comprimento máximo de outra lista de agrupamentos	Valor absoluto da diferença entre os comprimentos das listas	Profundidade máxima
30.880	50	22	71	49	3
23.160	25	42	68	26	3
23.160	50	42	67	25	2
23.160	100	42	67	25	2
18.528	50	85	38	47	2
15.440	50	142	26	116	2

Tabela 5.28: Forma final da RLC2 com as imagens de rosto, usando diferentes parâmetros.

Raio dos agrupamentos

Na busca de encontrar uma fórmula “universal” para o raio óptimo dos agrupamentos de nível zero da RLC2, fez-se uma tentativa de relacionar algumas das propriedades dos espaços métricos utilizados neste trabalho para se chegar ao valor definido como ideal para este parâmetro.

	Dicionário de alemão	Dicionário de inglês	Histogramas de imagens	Imagens de rosto
Média (μ)	11,74	8,35	0,43	30.203
Desvio padrão (σ)	3,10	3,90	0,17	7.867
Raio escolhido	10,00	6,00	0,71	23.160
μ - σ / 2	10,2	6,4	0,34	26.269

Tabela 5.29: Proposta de valores para o raio da RLC2.

A partir da informação da tabela 5.29, e exceptuando, mais uma vez, os histogramas de imagens, observa-se que os raios escolhidos são bastantes próximos dos valores dados pela fórmula μ - σ / 2. É claro que, a partir desta fórmula, não se encontra imediatamente o raio óptimo para cada domínio, mas obtém-se um bom ponto de partida para a descoberta do mesmo.

No que diz respeito aos histogramas de imagens, estamos claramente a lidar com uma distribuição diferente da dos outros domínios. É possível constatar esse facto a partir da informação presente no capítulo 4. Este foi o único domínio em que o raio utilizado durante os testes é superior à média das distâncias entre os seus pontos. Chegou-se a pensar que, por se tratar da maior base de dados, poderia existir alguma questão relacionada com a escalabilidade da RLC2. Mas, depois de se reduzir o número de inserções de um dos ficheiros de dados utilizados para cerca de metade, verificou-se que o valor obtido para o raio dos agrupamentos era bastante parecido. O mesmo aconteceu com a distribuição das distâncias deste novo espaço métrico. Portanto, a divergência de valores entre este espaço métrico e os restantes deve estar relacionada com as distribuições das distâncias dos pontos.

Capacidade das folhas

Relativamente à capacidade das folhas da RLC2, foi escolhido o valor 100 para os dicionários de palavras, 50 para os histogramas de imagens e 25 para as imagens de rosto. Apesar de este parâmetro não ter o mesmo impacto no desempenho da RLC2 como tem o

raio dos agrupamentos, os valores escolhidos apresentam alguma consistência, exceptuando novamente o caso dos histogramas de imagens.

Uma vez que a organização das folhas em memória secundária pode ser feita utilizando mais do que uma página, tem alguma lógica que os melhores desempenhos ocorram quando uma folha não ocupa demasiadas páginas. Se assim for, faz sentido que se utilizem capacidades maiores quando se guardam pontos mais pequenos e capacidades menores quando os pontos ocupam mais espaço.

Como cada palavra do dicionário de alemão ocupa 33 bytes, cada palavra do dicionário de inglês ocupa 21 bytes e cada imagem de rosto ocupa 192 bytes, é coerente que a capacidade das folhas seja menor quando se armazenam imagens de rosto do que quando se guardam palavras de um dicionário. Contudo, como cada histograma de imagem ocupa 896 bytes, esperava-se que a capacidade das folhas para este domínio fosse igual ou inferior a 25. Mas não é o caso.

6. Conclusões

Neste capítulo são apresentadas as conclusões finais deste trabalho e são indicadas direcções para trabalho futuro.

6.1 Apreciação Crítica do Trabalho Desenvolvido

Neste trabalho foi apresentada a Recursive Lists of Clusters 2, uma estrutura de dados métrica genérica, dinâmica e implementada em memória secundária. Esta estrutura é uma variante de outra estrutura de dados, a Recursive Lists of Clusters.

Para a concepção desta variante, foram realizadas as seguintes tarefas.

- Descrição de algumas estruturas de dados inseridas no mesmo âmbito que a RLC2, ou seja, estruturas de dados métricas genéricas, dinâmicas e implementadas em memória secundária. São os casos da M-tree, Slim-tree, DF-tree, SM-tree, D-Index e RLC.
- Familiarização com a implementação da RLC e realização de ajustes para que passasse a ser genérica.

Depois de desenvolvida a RLC2, as tarefas realizadas de modo a avaliar o seu desempenho foram as seguintes.

- Implementação da versão original da RLC, a RLC0.
- Escolha dos espaços métricos a utilizar nos testes. Foram escolhidos quatros espaços métricos de características diferentes cujas bases de dados são constituídas por dados reais.
- Familiarização com as implementações da M-tree, da Slim-tree e da DF-tree.

- Parametrização de todas as estruturas de dados disponíveis. Essa parametrização consistiu na consulta de bibliografia para algumas estruturas e na comparação de resultados experimentais para outras.
- Análise dos resultados experimentais. Foram realizados testes experimentais para comparar o desempenho de todas as estruturas de dados em relação a diferentes métricas. No confronto directo com a RLC, a M-tree, a Slim-tree e a DF-tree, a RLC2 mostrou-se consideravelmente mais eficiente no que diz respeito a operações de pesquisa por proximidade. Nas operações de inserção, a RLC2 demonstrou relativa superioridade em relação às outras estruturas, sendo seguida de perto pela Slim-tree. Em comparação com a RLC0 e, apesar dos bons desempenhos desta nas inserções, remoções e pesquisas por proximidade, a RLC2 ainda conseguiu obter melhores resultados.
- Estudo mais pormenorizado da RLC2. Neste estudo foram apresentadas as razões pelas quais o desempenho da RLC2 supera o da RLC e foi feita uma análise à forma final que a RLC2 apresenta com cada espaço métrico utilizado. Finalmente, tentou-se encontrar uma maneira para descobrir os melhores parâmetros da RLC2.

Apesar de todo este trabalho, uma das limitações da implementação da RLC2, tal como da implementação da RLC, continua a ser a impossibilidade de um objecto ocupar mais do que uma página. Mesmo com os resultados da RLC2, este é, sem dúvida, um problema que merece atenção.

Uma condicionante encontrada durante a elaboração desta dissertação foi o facto de não ter sido possível obter estruturas de dados que permitissem remoções de objectos. Este problema limitou a avaliação de desempenho da RLC2.

Mesmo nas implementações obtidas, verificaram-se algumas dificuldades. Os desempenhos da Slim-tree e da DF-tree podem ter saído prejudicados dos testes experimentais efectuados, devido à não implementação do algoritmo *Slim-down* na biblioteca *arboretum*.

Outra questão que também deve ser levantada está relacionada com o número reduzido de espaços métricos testados. Infelizmente, por causa do tempo dispendido na grande quantidade de testes experimentais efectuados, bem como do tempo que cada teste demora a ser realizado (alguns dos testes demoram cerca de quarenta e seis horas), não foi possível enriquecer ainda mais este trabalho com a utilização de mais espaços métricos.

6.2 Trabalho Futuro

Tal como referido anteriormente, a existência de um limite máximo para a dimensão de um objecto é uma questão que deveria merecer mais atenção. Seria desejável que essa limitação fosse eliminada da implementação.

Outro pormenor digno de futura investigação prende-se com a discrepância nos resultados dos testes entre o domínio dos histogramas de imagens e os restantes domínios. Pensa-se que as diferenças resultam do facto da distribuição das distâncias deste espaço métrico ser claramente diferente da dos outros. Claro que, para comprovar esta teoria, será necessário testar a RLC2 com outros espaços métricos que possuam distribuições semelhantes.

7. Bibliografia

[Ben75] J. L. Bentley. Multidimensional binary search trees used for associative searching. Communications of the ACM, 18(9): 509–517, 1975.

[Bri95] S. Brin. Near neighbor search in large metric spaces. Proceedings of the 21st International Conference on Very Large Data Bases (VLDB'95), páginas 574-584. Morgan Kaufmann Publishers, 1995.

[BO97] T. Bozkaya e M. Ozsoyoglu. Distance-based indexing for high-dimensional metric spaces. Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data, páginas 357-368. ACM Press, 1997.

[Com79] D. Comer. The Ubiquitous B-Tree, ACM Computing Surveys, 11(2):121–137, 1979.

[CNB+01] E. Chávez, G. Navarro, R. Baeza-Yates e J. L. Marroquín. Searching in metric spaces. ACM Computing Surveys, 33(3):273–321, 2001.

[CPZ97] P. Ciaccia, M. Patella e P. Zezula. M-tree: An efficient access method for similarity search in metric spaces. Proceedings of the 23rd International Conference on Very Large Data Bases (VLDB'97), páginas 426-435. Morgan Kaufmann Publishers, 1997.

[CP98] P. Ciaccia e M. Patella. Bulk loading the M-tree. Proceedings of the 9th Australian Database Conference (ADC 1998), páginas 15-26. Springer, 1998.

[Cha09] P. Chambel. Pesquisa de Imagens de Rosto. Dissertação de Mestrado em Engenharia Informática, Departamento de Informática, Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa, Quinta da Torre, 2829-516 Caparica, Portugal, 2009.

[DGS+03] V. Dohnal, C. Gennaro, P. Savino e P. Zezula. D-Index: Distance searching index for metric data sets. Multimedia Tools and Applications, 21(1):9-33, 2003.

[FB74] R. A. Finkel e J. L. Bentley. Quad trees: A data structure for retrieval of composite keys. Acta Informatica, 4(1):1–9, 1974.

[Gut84] A. Guttman. R-Trees: A dynamic index structure for spatial searching. Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data, páginas 47-57. ACM Press, 1984.

[Mam05] M. Mamede. Recursive lists of clusters: A dynamic data structure for range queries in metric spaces. Proceedings of the 20th International Symposium on Computer and Information Sciences (ISCIS 2005). Lecture Notes in Computer Science, 3733, páginas 843-853. Springer-Verlag, 2005.

[Mam07] M. Mamede. A Dynamic Data Structure for Range Queries in High Dimensional Metric Spaces. http://ctp.di.fct.unl.pt/~mm/dynamic-07.pdf, 2007.

[MB07] M. Mamede e F. Barbosa. Range queries in natural language dictionaries with recursive lists of clusters. Proceedings of the 22nd International Symposium on Computer and Information Sciences (ISCIS 2007), páginas 1-6. IEEE CS Press, 2007.

[Rod06] C. Rodrigues. Implementação de sistemas de indexação para espaços métricos. Relatório do Projecto Final de Curso, Departamento de Informática, Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa, Quinta da Torre, 2829-516 Caparica, Portugal, 2006.

[SS04a] A. P. Sexton e R. Swinbank. Bulk loading the M-tree to enhance query performance. Proceedings of the 21st Annual British National Conference on Databases (BNCOD 21). Lecture Notes in Computer Science, 3112, páginas 190-202. Springer-Verlag, 2004.

[SS04b] A. P. Sexton e R. Swinbank. Symmetric M-tree. Technical Report CSR-04-02, School of Computer Science, University of Birmingham, UK, 2004. Available at URL http://www.cs.bham.ac.uk/~rjs/research/ .

[STT+01] R. F. Santos Filho, A. Traina, C. Traina Jr. e C. Faloutsos. Similarity search without tears: The OMNI family of all-purpose access methods. Proceedings of the 17th International Conference on Data Engineering (ICDE'01), páginas 623-630. IEEE Computer Society, 2001.

[TZ06a] A. Thomasian e L. Zhang. Persistent semi-dynamic ordered partition index. The Computer Journal, 49(6):670-684, 2006.

[TZ06b] A. Thomasian e L. Zhang. The stepwise dimensionality increasing (SDI) index for high-dimensional data. The Computer Journal, 49(5):609-618, 2006.

[TTF+02] C. Traina Jr., A. Traina, C. Faloutsos e B. Seeger. Fast indexing and visualization of metric data sets using Slim-trees. IEEE Transactions on Knowledge and Data Engineering, 14(2):244-260, 2002.

[TTS+02] C. Traina Jr., A. Traina, R. Santos Filho e C. Faloutsos. How to improve the pruning ability of dynamic metric access methods. Proceedings of the 11th International Conference on Information and Knowledge Management (CIKM'02), páginas 219-226. ACM Press, 2002.

[TP91] M. Turk, A. Pentland. Face Recognition using eigenfaces. Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, páginas 586-591. 1991.

[Yia93] P. N. Yianilos. Data structures and algorithms for nearest neighbor search in general metric spaces. Proceedings of the 4th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'93), páginas 311-321. Society for Industrial and Applied Mathematics, 1993.

[ZAD+05] P. Zezula , G. Amato , V. Dohnal , M. Batko. Similarity Search: The Metric Space Approach. Springer-Verlag, 2005.

A. Anexos

A 1. Resultados Experimentais Usando Páginas de 4.096 Bytes

Uma vez que as limitações da implementação da RLC não permitiram que todos os testes se realizassem com páginas de 4.096 bytes, decidiu-se repetir os mesmos testes, para as restantes estruturas, com páginas dessa dimensão. Os testes do domínio das imagens de rosto não foram repetidos pois já tinham sido realizados com páginas de 4.096 bytes.

	M-tree	Slim-tree	DF-tree	RLC2
Nº médio de cálculos de distâncias entre dois pontos durante inserções	8.439	90	2.831	371
Nº médio de cálculos de distâncias entre dois pontos durante pesquisas	24.213	47.215	24.740	8.585
Nº médio de leituras durante inserções	3	3	34	6
Nº médio de leituras durante pesquisas	1.144	1.019	1.180	267
Nº médio de escritas durante inserções	4	4	5	2,5

Tabela A.1: Resultados dos testes, usando páginas de 4.096 bytes, com o dicionário de alemão.

	M-tree	Slim-tree	DF-tree	RLC2
Nº médio de cálculos de distâncias entre dois pontos durante inserções	13.245	100	4.798	250
Nº médio de cálculos de distâncias entre dois pontos durante pesquisas	26.786	48.531	31.935	10.533
Nº médio de leituras durante inserções	2,9	2,9	42	3,4
Nº médio de leituras durante pesquisas	893	771	931	447
Nº médio de escritas durante inserções	4	3,9	5	2,2

Tabela A.2: Resultados dos testes, usando páginas de 4.096 bytes, com o dicionário de inglês.

	M-tree	Slim-tree	DF-tree	RLC2
Nº médio de cálculos de distâncias entre dois pontos durante inserções	133	38	995	37
Nº médio de cálculos de distâncias entre dois pontos durante pesquisas	46.806	62.137	53.981	3.974
Nº médio de leituras durante inserções	16	11	307	15
Nº médio de leituras durante pesquisas	25.569	34.707	34.707	1.476
Nº médio de escritas durante inserções	20	13	14	9,7

Tabela A.3: Resultados dos testes, usando páginas de 4.096 bytes, com os histogramas de imagens.

A partir dos resultados presentes nas tabelas A.1, A.2 e A.3, e tendo em conta os resultados com páginas maiores, percebe-se que o desempenho de todas as estruturas piorou no que diz respeito a operações de acesso a disco. Desta vez, excluindo os resultados obtidos anteriormente para as imagens de rosto, a Slim-tree destaca-se pelo seu desempenho relativo ao número médio de leituras durante inserções. Relativamente às operações de pesquisa (número médio de cálculos de distâncias entre dois pontos e número médio de leituras), a RLC2 continua a mostrar grande superioridade em relação a todas as

outras. A RLC2 regista também, em todos os testes, os melhores resultados no número médio de escritas durante inserções.

Sendo assim, pode-se dizer que, entre estas estruturas, a RLC2 será a melhor para a execução de operações de pesquisa e rivaliza com a Slim-tree no que diz respeito a operações de inserção.